Interleaved Concatenated Coding for Input–Constrained Channels

A THESIS

Presented to

the Academic Faculty

By

Kofi D. Anim-Appiah

In Partial Fulfillment
of the Requirements for the Degree of
Doctor of Philosophy in Electrical Engineering

School of Electrical and Computer Engineering
Georgia Institute of Technology

November, 2000

Interleaved Concatenated Coding for Input-Constrained Channels

Approved:

Professor Steven W. McLaughlin, Chairman

Professor John R. Barry

Professor Gordon L. Stüber

Date approved by Chairman 10-9-00

for Naa Abadae — my "sweetness"

Acknowledgments

I am particularly delighted to be able to express my gratitude and thanks to Professor McLaughlin for the opportunity to study turbo codes and digital recording with him. His affability, constant encouragement, and eagerness to spend time with his graduate students have made these difficult years a lot more bearable. He has been much more than a first-rate academic advisor to me — a feeling I believe is shared by all of the graduate students fortunate enough to have him as advisor — and I sincerely wish him God's blessings in all his future endeavors.

I would also like to thank Professors Barry and Stüber for serving on my thesis reading committee, and also, Professors Mersereau and Bunimovich for agreeing to serve on the defense committee.

Throughout all the uncertain times, my wife Naa has been the model spouse — supporting me with prayers, encouragement, love, and more prayers. She has willingly shared the sacrifice of family time that was needed to complete this work. I am indebted to her. Naa, *wara na medɔ*.

I am also grateful for the emotional and spiritual support from my father, brother Desmond, and sisters Amy and Michelle. Their noticeable impatience with the length of time I was spending in school did much to hasten me through the perils of this program. And I also acknowledge my dear mother, Esther Harriet Abena Mankosa (1935–1990), whose spirit is often near me.

My initiation into, and understanding of the principles of "concatenating and iterating" was nurtured by numerous fruitful discussions with several fellow Ph.D.

students at the Georgia Institute of Technology. In particular I recall many enlightening discussions with Anh Nguyen, Laura McPheters, Ravi Sivasankaran, and Suparna Datta. I also enjoyed talking about Brazilian and jazz music with Renato Lopes, especially after the discussions on *a posteriori* probabilities had surprisingly begun to cloy.

Lastly, but by no means least, I thank and praise the Good Lord God for grace, mercy, a lifetime of blessings, and for salvation through his son Jesus Christ. He ultimately, has made this accomplishment possible.

Contents

List of Tables

List of Figures

Glossary of Symbols and Acronyms

E_b	$:=$	Energy per information bit
N_o	$:=$	Additive white Gaussian noise power spectral density
D	$:=$	Unit delay operator
$\boldsymbol{\pi}$	$:=$	Interleaver function
κ	$:=$	Block code dimension
k	$:=$	Maximum runlength parameter
η	$:=$	Constrained code efficiency
$C_{(d,k)}$	$:=$	Capacity of (d, k) constraint
g_{fb}	$:=$	Binary shift-register feedback tap polynomial
g_{ff}	$:=$	Binary shift-register feedforward tap polynomial
$d_H(\boldsymbol{a}, \boldsymbol{b})$	$:=$	Hamming distance between $\boldsymbol{a}$ and $\boldsymbol{b}$
d_{min}	$:=$	Minimum distance of code
$d^2_{E,min}$	$:=$	Minimum squared Euclidean distance
$d^{(i)}_j$	$:=$	The j^{th} bit of the data block corresponding to the i^{th} codeword of a block code
$c^{(i)}_j$	$:=$	The j^{th} bit of the i^{th} codeword of a block code
$L(d_j)$	$:=$	Log *a posteriori* probability ratio for bit d_j
$L^e_{ij}(d_k)$	$:=$	Extrinsic part of LAPP ratio for bit d_k passed from decoder i to j
X_S	$:=$	Constraint forbidding the occurrence of binary string S
$Q(\cdot)$	$:=$	Complementary Gaussian cumulative distribution function
P_b	$:=$	Probability of bit error
$\mathcal{C}_n$	$:=$	Odd single parity check code of length n

$\mathcal{C}_{ns}$	:=	Systematic $(0, n-1)$-constrained code of length n
$x^{(j)}$	:=	A string of x's of length j
$\boldsymbol{a}\|\boldsymbol{b}$	:=	The concatenation of row vectors $\boldsymbol{a}$ and $\boldsymbol{b}$
APP	:=	*a posteriori* probability
AWGN	:=	Additive white Gaussian noise
BCJR	:=	Bahl, Cocke, Jelinek, and Raviv
BER	:=	Bit error rate
BPSK	:=	Binary phase shift keying
ECC	:=	Error control (or correction) code
E^2PR4	:=	Doubly extended partial response class 4
EPR4	:=	Extended partial response class 4
FSTD	:=	Finite-state transition diagram
ISI	:=	Intersymbol interference
LAPP	:=	log APP
LDPC	:=	Low density parity check
LLR	:=	log likelihood ratio
MAP	:=	Maximum a posteriori probability
MFM	:=	Modified frequency modulation
ML	:=	Maximum likelihood
MRC	:=	Magnetic recording channel
NRZ	:=	Non return-to-zero
NRZI	:=	Non return-to-zero, inverted
PAM	:=	Pulse amplitude modulation
PR1	:=	Partial response class 1
PR4	:=	Partial response class 4
PRML	:=	Partial response with maximum likelihood

RHS	:=	Right hand side
RLL	:=	Runlength limited
RSC	:=	Recursive systematic convolutional
SCCC	:=	Serial concatenated convolutional code
SNR	:=	Signal-to-noise ratio
SOVA	:=	Soft-output Viterbi algorithm
SPC	:=	Single parity check

SUMMARY

The foundation for the application of the recently introduced interleaved concatenated codes (parallel and serial) on digital data storage channels is steadily developing. This effort is being driven by the ever-increasing need for greater data storage densities as more and more of the data that is encountered in everyday use is transmitted and archived in digital format. The powerful error correction capabilities of these interleaved concatenated codes have been demonstrated by many researchers on channel models for both magnetic and optical storage—a lot more so for the former. These data storage channels are input-constrained, requiring that the binary sequences written to the channel obey certain constraints. Often, the soft-input requirements of the iterative decoders for the interleaved codes are difficult to meet with most constrained code decoders because a majority of these constraint decoders produce only binary decisions. The chief goal of this study is to directly address the interface between soft-input decoders for the error control code, and the output of the decoder for a constrained code. System configurations that allow the error control decoder access to reliable soft input in the presence of a constrained code are presented for both the memoryless additive white Gaussian noise (AWGN) channel and some precoded partial response (PR) channels. This is done within coded systems of both interleaved parallel and serial concatenated codes. We show how to implement these systems, provide simulation results, and analyze some of their properties. We also highlight the properties of some block constrained codes that affect the error rate performance of the overall system—in

certain cases we show how these properties lead to the design of some good high-rate block constrained codes for use with iterative decoders. These block codes are matched to the channel (AWGN or precoded PR) in that the code properties are dictated by the constrained channel and its detector. We show that while the AWGN channel requires good Hamming distance properties from high-rate block constrained codes, precoded PR channels (ISI channels in general) detected with an *a posteriori* probability (APP) detector require codes containing significant *a priori* information about some of the code bits. In connection with the use of nonlinear block constrained codes, we show how to implement a soft-input, soft-output (SISO) decoder for a general nonlinear block code; we also compare the complexity of this computation to that for linear convolutional codes. Throughout the study, we focus on describing constrained-system configurations and the corresponding decoding techniques that allow a constrained code to be used with minimal accompanying rate loss. Some of these results are applicable to other digital communication systems that use both constrained codes and error-control codes. A constrained code is a good candidate for use in some systems with a data-derived timing signal. Systems that employ baseband line signaling with coding might be one such example.

CHAPTER I

INTRODUCTION

Since their introduction in 1993 by Berrou *et al.* [7], the class of parallel concatenated recursive systematic convolutional codes has continually engaged the attention of many researchers. These "turbo" codes continue to enjoy a lot of attention due to their remarkable performance at low signal-to-noise ratios (SNR) and to the seemingly limitless investigation into coding, modulation, and equalization their basic encoding/decoding structure inspires. Although classically constructed from convolutional component codes, turbo codes resemble random block codes and owe much of their considerable low-SNR effectiveness to the ease with which very large block sizes can be obtained. The introduction [7] was soon followed by the discovery by Benedetto *et al.* in [8] of the class of interleaved serial concatenated codes. While the discovery of this new coding structure was undoubtedly inspired by turbo codes, it possesses some distinct differences from the latter class in fundamental performance and design requirements. These differences are illuminated by an analysis of the error rate performance of the structure. We will often refer collectively to both classes of codes (and other hybrids of both of these classes of codes) as simply interleaved concatenated codes. A lot of the initial research effort engendered by the introduction [7] was focussed on analyses of the codes in order to better understand the source of their power. Notably, Benedetto and Montorsi in a series of papers [10], [11], [12], [13], Perez *et al.* [56], Robertson [58], and Hagenauer

et al. [31] have provided much of the initial basic understanding available today about turbo codes. An increasing volume of recent research effort in turbo codes is now addressing the applications of these codes on a variety of channels. Among the many kinds of channels for which their use is being considered are deep space communication channels [20], wireless mobile channels [42], and more recently, data storage channels for magnetic and optical recording [61], [64] and [68]. Each of these channels poses its own unique problems that have to be surmounted in order to make the use of turbo codes a reality. The power of turbo codes comes at the cost of high latency from the requisite large turbo code block sizes (in excess of 1000) for effective coding gain, and the computationally complex iterative algorithm currently used to decode them. Recent research has reported some progress in dealing with the complexity issue by using, for example, reduced complexity log domain decoders [59].

The proposed research is concerned with the application of interleaved concatenated coding on data storage channels. Data storage channels can make use of channel codes to achieve higher recorded bit densities. Such channels include hard disk drives, tape systems, and floppy drives—which are examples of magnetic based storage systems. Optical based storage systems such as the ubiquitous compact disc and related media like CD-ROM and the more recent DVD systems also make use of channel codes. Increased use of channel codes for error correction coding (ECC) coupled with other signal processing techniques, primarily PR signaling, have been partially responsible for the dramatic increases in the recording densities in today's hard disk systems [49], [52]. Before the advent of interleaved concatenated coding, two distinct approaches to ECC for the magnetic recording channel (MRC) had been proposed. The first was a suitably precoded PR channel coupled with a convolutional code selected for large free distance; this was based on the initial work of Wolf &

Ungerboeck [76]. Essentially the PR channel, whose intersymbol interference (ISI) introduces memory into channel output sequences, is "inverted" with the precoder so that codes that perform well on memoryless channels can be effectively applied. On the other hand, the approach of Karabed and Siegel [37] seeks trellis codes whose spectrum matches that of the non precoded PR channel. This approach essentially appeals to Shannon's "water-filling" theorem [17] and is intended to exploit rather than undo the frequency selectivity of the channel. This is the same principle on which discrete multitone modulation is based—the basis for the xDSL modems that are recently rising in popularity. Either of these two coding approaches provides about 3 dB of coding gain on PR type IV (PR4 or modified duobinary) digital recording channels. But recent results have indicated that interleaved concatenated codes applied to PR4 channels promise coding gains of about 5 dB [64], [68]. So there is naturally some excitement about the prospects for these newer codes in digital recording systems—the higher coding gains allow for higher recorded bit densities.

In most digital recording systems, system timing information is derived from the channel output sequences, therefore these sequences must possess self-clocking properties. This is ensured by constraining the channel input sequences; the translation of unconstrained sequences into constrained sequences being effected with a constrained code. The use of this code usually represents extra overhead on the overall system since the redundancy used to enforce the constraint does not function to enhance codeword distance properties. To be concrete, suppose the constrained code $\mathcal{C}$ is implemented with a binary code of rate κ/n, with $\kappa < n$. There is thus a reduction in the energy per constrained coded bit compared to that for the unconstrained case. Therefore a loss in performance of

$$l_{\mathcal{C}} = 10\log\left(\frac{n}{\kappa}\right) \text{ dB} \tag{1.1}$$

is usually observed with binary-input decoders. This is referred to as the nominal **rate loss** of the constrained code. Constrained codes therefore tend to have high rates in order to minimize the nominal rate loss.

An important issue in the implementation of interleaved concatenated coding on constrained channels is the interface between the constrained code and the ECC. The decoders for the interleaved concatenated codes require soft inputs—they are soft-input soft-output (SISO) decoders. In the standard recording system configuration the constrained code is downstream (closer to the channel) of the ECC and so the input to the ECC decoder is obtained from the constrained code decoder (see Figure 2.1). The typical constrained code decoder is binary input binary output; obtaining faithful soft information from many popular constrained code decoders can therefore be very difficult since the high-rate encoders that minimize $l_{\mathcal{C}}$ often obscure the relationship between encoder input and output bits. In some cases the constrained code can be viewed as an equivalent block code; for these cases it is then possible to contemplate a SISO decoder as we later show in this thesis. The practical implementability of a SISO decoder for a block code, though, is limited to codes with relatively small cardinality (perhaps fewer than 2^{10} codewords). This issue of requiring soft information from constraint decoders ('soft demodulation') has hitherto not been a problem because the decoders for the ECCs that have been employed on constrained channels are hard- or binary-input type. So the constrained code decoder simply passes binary decisions to the ECC decoder without any difficulty.

Requiring soft outputs from constrained code decoders, can frequently result in the condition

$$l_{\mathcal{C}} > 10\log\left(\frac{n}{\kappa}\right) \text{ dB} \tag{1.2}$$

if those soft outputs are not of sufficient fidelity. This is the case when it is difficult to ascertain the exact relationship between encoder input and output bits, or if the code and encoder are simply not well-suited for faithful generation of soft information. Either way, this more severe condition is a result of severe error propagation through the constrained code decoder and is tantamount to the error rate at the constrained decoder output being worse than that at the encoder input.

1.1 Problem and Solution

Most recent work on interleaved concatenated coding for the magnetic recording channel (MRC) has focussed on high-rate concatenated codes and PR signaling [22], [23], [51], [61], [64], and [68]. These papers have established the suitability of high-rate concatenated codes for PR-equalized channels. Of these authors, only Fan [22] and Fan & Cioffi [23] have specifically addressed interleaved concatenated codes on constrained channels. In this thesis we are primarily concerned with investigating ways of employing interleaved concatenated coding on constrained channels of which the MRC will be used as the main example. Several approaches are taken. The most general view is one of proposing and evaluating different recording system configurations that allow the SISO decoders for the interleaved concatenated codes to obtain reliable soft information so that the full potential of the overall system can be realized. Underpinning this investigation is the goal of minimizing the rate loss associated with the use of the constrained code. It is also a goal to describe and evaluate systems that allow us to recover some of the rate loss of the constrained code at the expense of increased receiver complexity. A lower level view of the scope covers system design and evaluation through simulation and analysis, methods of obtaining soft information from decoders for some constrained codes, block code

design for constrained codes, and good constrained codes for specific channels. As far as channels are concerned, the thesis focusses on constrained AWGN and noisy PR channels that are widely used for magnetic recording. There is heavy emphasis on the class of $(0, k)$ constraints. We now provide brief, but somewhat more detailed descriptions of the proposed techniques and the rationale for considering them.

1.1.1 $(0, k)$-Constrained-Input Turbo Codes

The standard recording system is characterized by an outer ECC and an inner constrained code. This configuration is not well-suited for cases where a general constrained code is to be used with an ECC which has a soft-input decoder. This is because it is generally not known how to obtain soft information from most constrained code decoders. Turbo codes make use of an overall systematic encoder (Figure 2.7). The action of the encoder is therefore equivalent to periodically inserting parity bits into the systematic stream. If the systematic encoder input is (d, k)-constrained, then the output of the systematic encoder will be $(0, k')$-constrained ($k' > k$), the nonzero d parameter being converted into $d = 0$. Therefore we will only consider encoder input sequences that are $(0, k)$-constrained.

This 'reverse concatenation' allows the soft-input decoder to obtain reliable soft information directly from the channel detector. It can then pass corrected hard decisions to the constraint decoder. Since the input to one of the constituent RSC encoders within the turbo encoder is $(0, k)$-constrained we would like to provide the APP decoder for that RSC encoder with maximal information about the structure of the sequences at its input. This should improve the error rate for the encoder input since the decoder has *a priori* knowledge about the structure of the encoder input sequences. An effective way to do this is to perform APP decoding for that RSC

encoder over a Cartesian product trellis. The Cartesian product is that of the $(0, k)$ state and the RSC encoder state.

This constrained-input system and decoding technique are proposed for effectively recovering some portion of the rate loss that will be suffered by use of the constrained code. Since there is increased complexity in one APP decoder associated with the Cartesian product the tradeoff between increased complexity and error rate performance is investigated.

1.1.2 Performance of High-Rate $(0, k)$ Block Codes

For the standard recording system we propose to implement a $(0, k)$-constrained code as a short high-rate block code. It can readily be shown that rate $(n-1)/n$ $(0, k)$-constrained block codes can be constructed. In this research, the error rate performance of short high-rate $(0, k)$-constrained block codes is analyzed, and the performance limits of these codes when decoded with an ML decoder in AWGN is determined. The performance limits are examined by a detailed investigation of code and encoder construction for some families of rate $(n-1)/n$ $(0, k)$-constrained block codes. A union bound is developed for evaluation of the encoder input BER achieved by an ML decoder; this bound is then used to explain the performance capabilities of high-rate block codes with an ML decoder in AWGN channel. This performance is then claimed for a maximum *a posteriori* (MAP) decoder since such a decoder has to be at least as good as an ML decoder.

1.1.3 Turbo Codes Cascaded with High-Rate Block Codes on $(0, k)$-Constrained Channels

A SISO algorithm for a general block code is proposed. Considering the standard recording system configuration of an outer ECC/inner constrained code, this SISO

algorithm is proposed for soft demodulation of a short high-rate $(0, k)$-constrained block code. This soft information is then passed to the soft-input decoder for a turbo code. The performance of this system is simulated for several high-rate $(0, k)$-constrained block codes. The use of systematic $(0, k)$ modulation codes for PR (ISI in general) channels is also proposed; this is for the case where the channel detector is an APP detector like the BCJR algorithm or SOVA. The fixed coordinates of these systematic codes, although not contributing any additional distance to the code, are shown to be effective in improving the performance of an APP algorithm permitting a portion of the rate loss of the code to be recovered on PR channels.

1.1.4 (d, k)-Constrained Serial Concatenation

Several systems for incorporating (d, k) codes into an interleaved serial concatenation system are proposed. These systems involve systematic modulation codes and several examples of finite-state constrained encoders. Many conceptually viable (d, k)-constrained serial concatenation systems are compared in light of some of the results from earlier chapters. A system that makes use of systematic $(0, k)$ modulation codes and fully exploits the fixed coordinates of these codes to minimize the rate loss of the modulation code is proposed and noted for its simplicity and effectiveness—assuming the relatively large k parameter for such $(0, k)$ modulation codes can be tolerated by the system. A general way to employ a suitable precoder to obtain interleaver gain for rate p/q modulation codes with finite-state (i.e. trellis) encoders is also described.

1.1.5 Thesis Outline

This thesis is outlined as follows. Chapter 2 covers some background material on magnetic recording, PR signaling, constrained codes, and interleaved parallel and serial concatenated codes. Chapter 3 discusses constrained-input turbo codes as a means of rate loss recovery in exchange for increased complexity. The potential of this scheme with some interesting subsets of $(0, k)$-constraints is also highlighted. A comparison to a recently proposed scheme is also drawn. In Chapter 4, we address the performance of high-rate (rate $(n-1)/n$) block $(0, k)$-constrained codes in AWGN. The discussion is introduced with two motivating examples. We then derive the union bound for nonlinear block codes and then we show how to construct several families of rate $(n-1)/n$ block $(0, k)$-constrained codes and show, using the union bound, that a tradeoff between k and rate loss recovery exists. Chapter 5 addresses the performance of an outer turbo code cascaded with an inner constrained code on both the AWGN and PR channels. A method for correctly computing LAPP values for the input bits of a general block code is shown. The complexity of the algorithm is also given and compared with that for the case where a trellis exists. Chapter 6 addresses several schemes for combining interleaved serial concatenation with (d, k)-constrained codes. Finally, Chapter 7 summarizes the key results in this thesis and suggests directions for future research on this topic.

CHAPTER II

BACKGROUND

2.1 Magnetic Recording

Current magnetic recording systems employ PR signaling with Viterbi detection to achieve recorded bit densities upwards of 5–10 Gbits/in^2 at data rates in excess of 1 Gbits/s, and current system designers do not feel that 500 Gbits/in^2 is beyond reach [34], [52]. Essentially a channel equalizer is used drive the natural system response to some predefined PR target and then a Viterbi detector based on the trellis of that PR target is used to detect the equalized channel. This technique is commonly referred to as PRML (Partial Response with Maximum Likelihood sequence detection) and has been in widespread use since 1990 or thereabouts. We will discuss the recording system and channel model first, followed by PR signaling.

2.1.1 Channel Model for the Read/Write Process

Digital data is stored on magnetic media using saturation recording where the medium is polarized in one magnetic state to represent a *one* and in the other state to represent a *zero*. Figure 2.1 shows a block diagram of a typical data storage system. First the user data is encoded by an ECC to protect against burst errors. Currently this ECC is usually an interleaved Reed-Solomon code. Next the encoded bits are encoded by a modulation (or constrained) code to yield the *channel* bits

which are the bits that are actually stored on the medium. The sequence of these channel bits is often required to satisfy some constraint which facilitates proper system operation, and the modulation code is used to produce this constrained sequence.

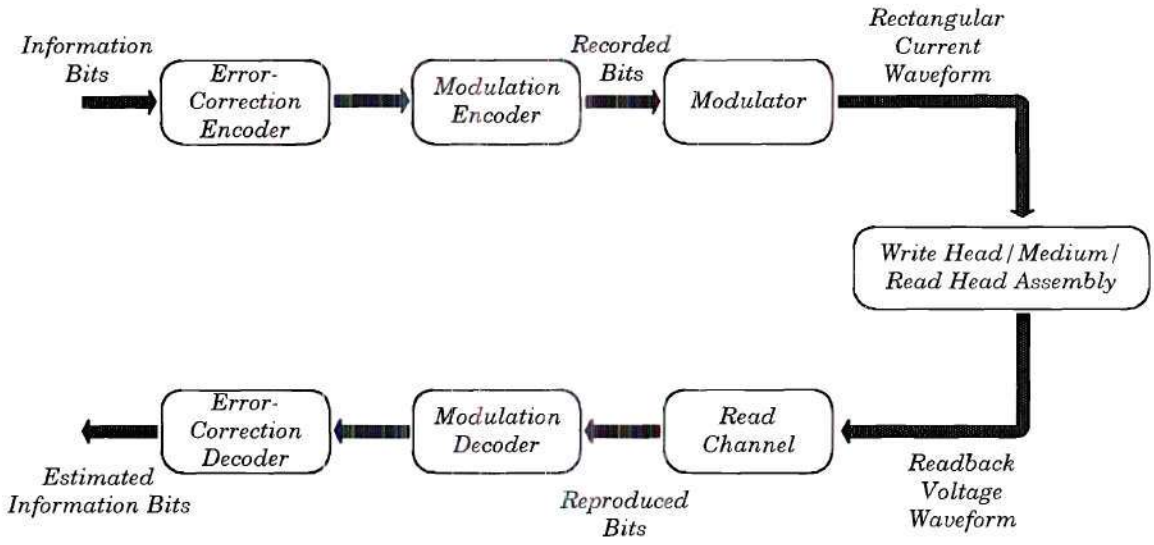


Figure 2.1. Block diagram of a data storage system.

Because the modulation code has a rate which is less than 1, each channel bit represents only a fraction of an information (or user) bit. The sequence of channel bits is then converted into either an NRZ or NRZI[1] rectangular current waveform which the write head then uses to polarize the medium in one of two states, effectively storing the channel bits; this is the write process. During the read process, the read head output is a continuous-time voltage waveform which is then amplified and passed through the read channel. The read channel consists of a

[1]NRZI is equivalent to precoding the binary data with $1/(1 \oplus D)$ and then using NRZ.

bandlimited filter, sampler, an equalizer, and a symbol detector. Finally, the modulation and then ECC decoders are applied. We next discuss a mathematical model of the continuous-time channel, which relates the readback signal to the recorded channel bits in Figure 2.1.

Though highly nonlinear, the recording channel can be modeled at a high level as a linear ISI channel with additive Gaussian noise, subject to a binary input constraint. A synchronous storage system in which channel bits occur at the fixed rate of $1/T_c$ channel bits/s is often assumed; T_c is thus a channel bit duration. The normalized input signal applied to the write head can be viewed as a two-level waveform which assumes the values $+1$ and -1 over consecutive time intervals of duration T_c, i.e.,

$$w(t) = \sum_i w_i p(t - iT_c) \tag{2.1}$$

with $p(t) = 1$ for $t \in [0, T_c]$. With NRZI line coding, the *transitions* of this waveform carry the digital information (channel bits) and are therefore constrained to occur at integer multiples of T_c; with NRZ line coding the *level* of the waveform carries the digital information. This waveform can be represented digitally as a sequence $\boldsymbol{w} = w_0 w_1 w_2 \cdots$ over the bipolar alphabet $\Phi = \{+1, -1\}$, where w_i represents the signal amplitude during the time period $(iT_c, (i+1)T_c]$. The continuous time readback waveform $y(t)$ generated by $w(t)$ is given by

$$y(t) = \sum_i (w_i - w_{i-1}) s(t - iT_c). \tag{2.2}$$

The "derivative" sequence $\boldsymbol{w}'$ with coefficients $w'_i = w_i - w_{i-1}$ consists of elements taken from the ternary alphabet $\{0, \pm 2\}$; the nonzero values corresponding to the transitions in the input signal alternate in sign. This property may be viewed as the memory of the channel and in practical systems is exploited by a sequence detector

to achieve a 3 dB gain over the simple peak detection method for which $\widehat{w} = |\boldsymbol{w}'|/2$. So the readback signal corresponds to a linear superposition of time-shifted versions of the transition response $s(t)$. Equivalently, viewing the read/write process as a digital Pulse Amplitude Modulation (PAM) system, the readback waveform may be written,

$$\begin{aligned} y(t) &= \sum_i w_i[s(t - iT_c) - s(t - (i+1)T_c)]. \\ &= \sum_i w_i h(t - iT_c) \end{aligned} \tag{2.3}$$

so that the effective impulse response of the channel is $h(t) = s(t) - s(t - T_c)$. This effective impulse response $h(t)$ is often referred to as the channel pulse response since it represents the head/medium response to an isolated current pulse. A frequently used model for the transition (or step) response $s(t)$ is the Lorentzian pulse

$$s(t) = \frac{1}{1 + (2t/\tau)^2}. \tag{2.4}$$

τ, often also denoted $PW50$ in the literature, gives the width of the pulse measured at 50% of its maximum height. Channel bit densities are often given as the ratio τ/T_c, with typical values of 2–2.5 or so. The simplest model for channel noise $n(t)$ assumes a white Gaussian process so that the read back signal is of the form

$$r(t) = y(t) + n(t), \quad n(t) \sim \mathcal{N}(0, \sigma^2) \tag{2.5}$$

There are other more realistic models that account for the effects of correlated media noise, material defects, and data dependence, but we do not consider any of those models.

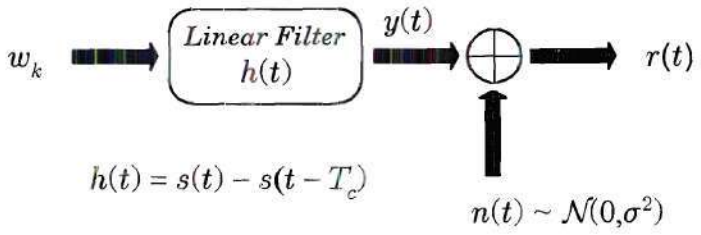


Figure 2.2. PAM equivalent of magnetic recording read/write process.

2.1.2 Partial Response Signaling

In PRML systems, a minimum mean square error (MMSE) equalizer is used to drive the discrete-time response of the channel to a 'target' PR polynomial; a Viterbi detector for the PR polynomial is then used for channel detection as shown in Figure 2.3. In this thesis, we will often make use of a soft-output algorithm operating on the trellis of the precoded PR target.

PR signaling was introduced by Lender in [44] but the classification of PR systems in use today (PR1, PR2, PR4, duobinary, etc.) was first introduced by Kretzmer [43]. Kabal and Pasupathy subsequently provided a unified study of these systems in [36] tabulating their properties according to Kretzmer's classification.

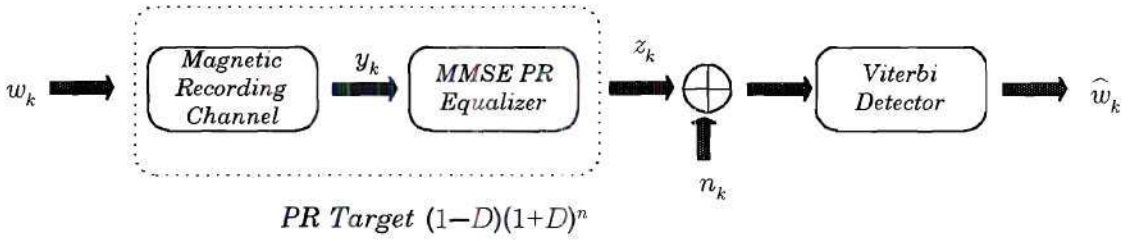


Figure 2.3. Equivalent discrete-time channel model of the PR-Equalized MRC.

Kobayashi and Tang were the first to recognize, in [41], the applicability of PR signaling to the MRC and proposed using an equalized channel response of $1 - D^2$

(D being the delay operator) with the then-current peak detection systems. For the equivalent discrete-time model shown in Figure 2.3, this PR4-equalized response results in

$$z_k = w_k - w_{k-2} \tag{2.6}$$

The $1 - D$ factor accounts for the read channel differentiation that naturally occurs due to the physics of saturation recording (the channel does not pass DC) while the $1 + D$ factor accounts for the ISI accompanying moderate-to-high density recording. One benefit of PR signaling is that it enables symbol transmission at a rate of $2W$ symbols s^{-1} (the Nyquist rate) on a bandlimited channel of bandwidth W with controllable ISI. Another benefit is that it can also be used to shape the spectrum of readback signals by placing (multiple) nulls at DC or the Nyquist frequency (one-half the symbol frequency). Some PR systems also place tracking-assisting pilot tones at these frequency nulls so that they interfere minimally with the data.

Shortly after the publication of [41], Kobayashi [40] suggested that Viterbi detection of the PR signal would recover the performance loss (3 dB) that is inherent in the reduced noise margin that accompanies the increased number of signal levels at the PR-equalized channel output when signaling with the $1 - D^2$ PR target. Not much later, Forney [26] showed that Viterbi detection was indeed optimal for ISI channels. Thapar and Patel [69] subsequently suggested the general class of PR polynomials $P_n(D) = (1 - D)(1 + D)^n$ for increasing the density of recorded bits on the basis that it provides a good match to the sampled response of the channel at different recording densities. For $n = 1$, Kobayashi [40] first observed that $P_1(D)$ corresponds to two independent interleaved $1 - D$ (dicode) channels. It can therefore be detected with a pair of Viterbi detectors for the dicode channel after the 'global' sample stream has been demultiplexed into 2 independent streams. This parallelism

offers advantages in greater detection speed and simpler decoders. In the recent past, most current systems have used $n = 1$ (modified duobinary — or PR4) but systems based on higher order polynomials, particularly $n = 2$ and $n = 3$ (EPR4, and E^{2}PR4, respectively), are becoming more common and are receiving more attention in the research literature [38]. Primarily for the purpose of containing error propagation, PR systems are often implemented with a precoder which attempts to invert $P_n(D)$ *modulo*-2 (for binary input channels). A generalized precoder then for $P_n(D)$ with binary input restriction is thus

$$\begin{aligned} P_n^{inv}(D) &= \frac{1}{(1-D)(1+D)^n} \bmod 2 \\ &= \frac{1}{(1-D)(1+D)^n \bmod 2} \\ &= \frac{1}{(1 \oplus D)(1 \oplus D)^n} \\ &= \frac{1}{(1 \oplus D)^{n+1}} \end{aligned} \tag{2.7}$$

with the symbol $\oplus$ denoting *modulo*-2 (XOR) addition. This is the Tomlinson-Harashima precoder for this channel [27]. The resulting precoded PR trellis still retains memory but its output can be detected on a symbol-by-symbol basis. Other kinds of precoders are also used with $P_n(D)$ — in particular [62] studies the effects of different precoders on the performance of serial concatenated systems in which the inner code is a precoded PR target. It is also the case that the precoded PR4 trellis is also equivalent to a pair of independent precoded dicode channels, therefore, in this thesis, we only consider this precoded dicode channel in any evaluation of precoded PR4 systems. Figure 2.4 shows the trellises for the dicode and precoded dicode channels, respectively.

Currently PR signaling is not widely used on optical recording channels but when it does, the general polynomials will almost certainly be of the form

$P_{n_o}(D) = (1 + D)^n$. This is because the channel does not differentiate (it passes DC) and therefore the $(1 - D)$ factor is unnecessary.

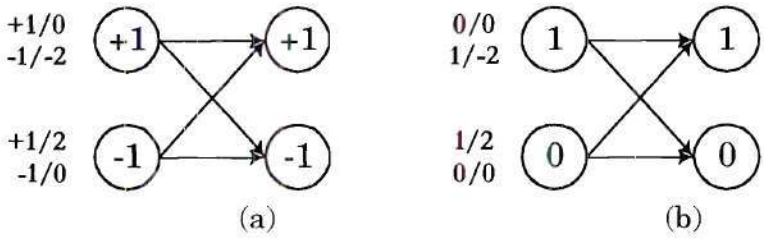


Figure 2.4. Trellises. (a) Dicode. (b) Precoded dicode.

Since $P_{n_o}(D)$, like $P_n(D)$, is also governed by a catastrophic trellis (single trellis output bit error can result in an infinite number of trellis input bit errors), it requires an appropriate precoder to contain error propagation; it is easy to see that the recursive $(1 \oplus D)^{-n}$ can be used. Currently, these channels mostly use peak detection methods just as was used in the early days of magnetic recording [49].

2.2 Constrained Codes

Optical and magnetic recording channels are often referred to as constrained channels because of the requirement that the sequences to be written to the channel obey certain constraints. On PR4-equalized channels, due to the action of the precoder (2.7), any runlength constraints imposed on the precoder input are directly transferred to the sampled readback sequences. This is the reason why the constraints can be imposed on the precoder input sequences — the constraints are actually required for samples r_i of the readback waveform $r(t)$. These constraints are imposed for a variety of purposes. They may be needed to ensure proper operation of some system functions like timing recovery and may also be necessary to alleviate some problems associated with high density recording on the media. Constraints are

often imposed with a recording code (also referred to as a modulation code) which translates unconstrained sequences into constrained sequences governed by the constraint. Examples of such constrained codes are (d,k)-constrained codes [47], One-Pairs (OP) codes [50], and the Maximum Transition Run (MTR) codes of Moon and Brickner [53]. MTR and OP codes are actually specialized (d,k) codes, used respectively, to mitigate the effects of magnetic interactions in the media (also provides coding gain), and to improve timing recovery in PRML systems. For PRML detected channels, constraints can also be used to improve the raw bit error rate off the storage medium by increasing the Euclidean distance between allowable sequences at the channel output. Karabed and Siegel have discussed this in [38] and [39].

This thesis concentrates on the family of binary (d,k) constraints where successive *one*s in the constrained sequence are separated by at least d and at most k *zero*s ($d < k$). The need for these constraints has significant consequences when interleaved concatenated codes are considered for use on optical and magnetic recording channels. The family of (d,k) constraints forms the basis for the most commonly used constrained codes on recording channels. These constrained codes are used to control various aspects of the operation of the recording channel. The d parameter may be used to mitigate the effects of ISI while the k parameter is used to ensure the occurrence of sufficient transitions in the readback sequence to enable extraction of timing information. Since PR4 systems use a pair of Viterbi detectors for detecting the channel they often only require $(0, G/I)$ constraints since the Viterbi detector is inherently optimal for ISI channels [26]. G is the k parameter for the 'global' sequence, while I represents the same for the even and odd interleaved sub-sequences, respectively, of the global sequence. The precoded dicode channel supports quasi-catastrophic error events in that the trellis (Figure 2.4(b)) contains

semi-infinite paths beginning at the same state that correspond to sequences with Euclidean distance strictly less than the free distance of the trellis. Viterbi detection on this trellis would require infinite path memory to preserve its Euclidean distance. The $(0, G/I)$ constraints allow finite path memory to be used by forcing any such two paths to remerge in finite time, thus preserving the Euclidean distance of the trellis [37]. The precoded PR1 trellis, on the other hand, does not have this problem.

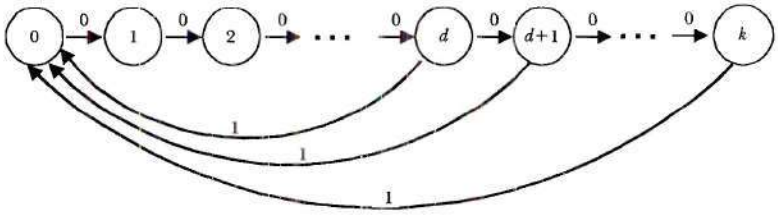


Figure 2.5. FSTD for binary (d, k) constraints

The family of binary (d, k) constraints is governed by the finite state transition diagram (FSTD) of Figure 2.5; the state designation is the number of *zeros* since the last *one*. Shannon [66] showed that the rate of a code that encodes into the constraint cannot exceed the capacity $C_{(d,k)}$ of the constraint, which is given as

$$C_{(d,k)} = \lim_{n\to\infty} \left\lceil \frac{\log_2 N_{(d,k)}(n)}{n} \right\rceil = \log_2 \lambda_{max} \tag{2.8}$$

with $N_{(d,k)}(n)$ being the number of sequences of length n obeying the constraint and λ_{max} the largest real eigenvalue of the corresponding $(k+1) \times (k+1)$ adjacency matrix $\boldsymbol{T}$ whose components $t_{i,j}$ are the number of ways of getting from state i to state j in one step. If the code has rate $R = p/q \leq C_{(d,k)}$, where unconstrained p-blocks are mapped to constrained q-blocks (the constraint is honored across q-block boundaries), the efficiency of the code is the ratio

$$\eta = \frac{R}{C_{(d,k)}}. \tag{2.9}$$

Closely related to (d,k)-constrained sequences are runlength limited (RLL) sequences for which the runlength of *like* symbols is at most k. The latter can be generated from the former by employing the recursive $(1 \oplus D)^{-1}$ precoder which converts a (d,k)-constrained sequence into a $(0, k+1)$ RLL sequence.

Table 2.1. Capacity $C_{(d,k)}$ of some (d,k) constraints. ([33]).

k	$d=0$	$d=1$	$d=2$	$d=3$	$d=4$	$d=5$	$d=6$
2	.8791	.4057					
3	.9468	.5515	.2878				
4	.9752	.6174	.4057	.2232			
5	.9881	.6509	.4650	.3218	.1823		
6	.9942	.6690	.4979	.3746	.2669	.1542	
7	.9971	.6793	.5174	.4057	.3142	.2281	.1335
8	.9986	.6853	.5293	.4251	.3432	.2709	.1993
9	.9993	.6888	.5369	.4376	.3620	.2979	.2382
10	.9996	.6909	.5418	.4460	.3746	.3158	.2633
11	.9998	.6922	.5450	.4516	.3833	.3282	.2804
12	.9999	.6930	.5471	.4555	.3894	.3369	.2924
13	.9999	.6935	.5485	.4583	.3937	.3432	.3011
14	.9999	.6938	.5495	.4602	.3968	.3478	.3074
15	.9999	.6939	.5501	.4615	.3991	.3513	.3122
∞	1.000	.6942	.5515	.4650	.4057	.3620	.3282

While short simple low rate constrained codes are often implemented as a block code with freely concatenable constrained codewords, high efficiency codes discourage this due to the sheer size of the codebook which must, of necessity, be comprised of long codewords. Typical rates for $(0,k)$-constrained codes are of the form $(n-1)/n$ with $n \geq 5$. On the other hand $(d \geq 1, k)$-constrained codes have much lower capacities and therefore code rates tend to be much lower (see Table 2.1).

Constrained code design falls generally into 3 broad classes. The state splitting algorithm of Adler, Coppersmith, and Hassner [1] provides a powerful, essentially systematic way of constructing codes with state-dependent encoders and sliding-block decoders at arbitrary rational rates below $C_{(d,k)}$. A simple well-known example of such a code is the Modified Frequency Modulation (MFM) rate $1/2$ $(1,3)$ code for which the two-state encoder is shown in Figure 2.6. This simple code has high efficiency of $\eta = .5/.5515 \simeq 91\%$ and can (among other ways) easily be decoded on a codeword-by-codeword basis since it is a systematic code.

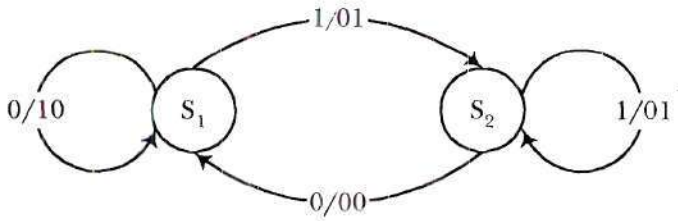


Figure 2.6. Finite-state MFM encoder.

Another general method falls under the heading of enumerative encoding/ decoding of permutation codes in which mixed radix numbering systems are used to enumerate the desired constrained codewords. It can be used to generate codes with very long codewords (high efficiency) which asymptotically approach capacity. Recent work by Datta and McLaughlin [18] typifies this technique; they have applied it to nonbinary (d,k) sequences. Lastly, there are also many good (d,k)-constrained codes designed in more or less *ad hoc* fashion by sheer ingenuity. Well-known examples of some constrained codes appear in [21], [46], and [70].

We now present an overview of the state splitting algorithm followed by a brief discussion of some ideas from Datta and McLaughlin [18]. These ideas will be used in the evaluation of some of the proposed systems in this thesis.

2.2.1 The State-Splitting Algorithm

This algorithm is used to produce a finite-state encoder which encodes unconstrained p-blocks into constrained q-blocks. The FSTD, G with $(k+1) \times (k+1)$ adjacency matrix $\boldsymbol{T}$ is the starting point for the code construction procedure. Pairs of states are connected by edges E which are labeled with the allowed code symbol that occurs when making a transition from state $b(E)$ to $e(E)$, which are the beginning and ending states, respectively, of E. The q^{th} power of G, G^q, is the FSTD whose i^{th} edge, E_i, is labeled with all the permissible q-blocks that exist in making a transition from state $b(E_i)$ to $e(E_i)$ in q steps along G; its adjacency matrix is T^q. p and q are first chosen such that $p/q < C_{(d,k)}$. A non negative approximate eigenvector (AE) $v = (v_0, ..., v_{k-1})^T$ is then found such that

$$\boldsymbol{T}^q v \geq 2^p \boldsymbol{v} \tag{2.10}$$

is satisfied. Franaczek [28] provides an algorithm (that also appears in [47]) for finding a v with the smallest maximal element since the number of state-splitting steps is upper bounded by $\sum_i (v_i - 1)$ and the sliding block decoder complexity is also strongly dependent on the maximal v_i. The existence of this AE is guaranteed by the Perron-Frobenius theory of non negative integer matrices [65]. This inequality ensures that, after the process of state splittings and mergings on G^q is completed, there will be at least 2^p edges leaving each state to allow unambiguous coding into the constraint (excess edges are deleted from the final encoder graph). If $v_i = 0$, state i (a 'weightless' state) and all edges emanating from or ending on it are deleted from G^q. A state is split by replacing it with two 'daughter' states each of which has as incoming edges all those of its parents. The outgoing edges of the parent are partitioned into 2 disjoint sets, one set assigned to each of the two daughters. Associated with this new FSTD $G^{q'}$ is an adjacency matrix $\boldsymbol{T}^{q'}$ and the goal of the

algorithm is to eventually produce, by a sequence of successive splittings, a final $G^{q'}$ for which the associated $\boldsymbol{T}^{q'}$ has approximate eigenvector consisting of only 0's or 1's. Lastly, all weightless states are removed. In an attempt to simplify the final encoder, two states can sometimes be merged to reduce the total number of states. This is possible when both states have the same edge labels. Further details of this procedure are given in [47]. The complexity of this encoder escalates with increasing block lengths and can quickly become unwieldy. While the algorithm is essentially mechanical, judicious selection of the sequence of splittings and mergings is rewarded with a simpler final decoder and shorter design time. The finite-state encoder just described above can be decoded with a sliding block decoder. This decoder, while state-independent, does require some look ahead (employing a look up table) for unambiguous decoding of a block of constrained bits; the number of required look-ahead bits increases with the number of rounds of state splittings used in designing the encoder. The advantage of the state-independent decoding though, is that error propagation is contained to within one p-block. McLaughlin [48] gives some examples of M-ary codes designed using this procedure.

2.2.2 Ideas from Enumerative Permutation Codes

Wolf [75] suggested using the permutation codes of Slepian [67] to construct high-rate runlength limited codes; Datta and McLaughlin [18] have recently showed one such construction that encodes and decodes M-ary (d, k)-constrained permutation codes with low complexity. The allowable sequences in a binary (d, k)-constrained code are comprised of *phrases* which are strings of at least d and at most k *zeros* terminating in a single *one*. These *phrases* are freely concatenable and therefore, the set of $k - d + 1$ *phrases*

$$\{0^{(d)}1, 0^{(d+1)}1, ..., 0^{(k-1)}1, 0^{(k)}1\}$$

where $0^{(x)}$ represents a string of x *zeros*, embodies the family of (d, k) constraints. As an example, we can parse the following $(0, 4)$-constrained sequence into its constituent *phrases* as given below;

$$\boldsymbol{y} = \underline{001}\ \underline{1}\ \underline{01}\ \underline{00001}\ \underline{0001}$$

Enumerative permutation encoding can be used to construct high-rate fixed-length (d, k)-constrained codewords by using permutations of a particular discrete distribution of these *phrase* lengths. Zehavi and Wolf [77] show that this distribution is such that a code achieving maximum information rate has the following properties;

(i) The *phrase* lengths L_i are statistically independent and identically distributed random variables;

(ii) The probability distribution of L is according to

$$P_L(l) = 2^{-lC},\ l = d + 1, ..., k + 1 \tag{2.11}$$

Any (d, k)-constrained code that achieves capacity satisfies (i) and (ii), and conversely, any code satisfying (i) and (ii) achieves capacity.

Code construction begins by specifying a *phrase* profile vector, $\boldsymbol{n} = [n_1, n_2, ..., n_N]$ with $N = k - d + 1$ distinct *phrases* and in which there are n_i occurrences of the i^{th} *phrase*. The codebook $\mathcal{C}$ has cardinality

$$|\mathcal{C}| = \binom{\sum_i n_i}{n_1,\ n_2, \cdots, n_N} \tag{2.12}$$

which is the total number of distinct permutations of the vector

$$\boldsymbol{l} = [\underbrace{l_1, ..., l_1}_{n_1}, \underbrace{l_2, ..., l_2}_{n_2}, ..., \underbrace{l_N, ..., l_N}_{n_N}].$$

There is then a one-to-one correspondence between each distinct permutation of l, and a (d,k)-constrained sequence of length $Q = \sum_{j=d+1}^{k+1} j n_j$. The rate of this code is then $R = \log_2|\mathcal{C}|/Q$. The n_i are chosen to satisfy

$$\frac{n_i}{\sum_j n_j} \approx 2^{-L_i C_{(d,k)}} \tag{2.13}$$

Clearly, the larger N is (implying longer codewords) the better this approximation is, and capacity is reached asymptotically. We do not go into the algorithmic details of encoding and decoding; they can be found in [18].

2.3 Interleaved Concatenated Codes

Code concatenation, originally introduced by Forney [25], is used to obtain composite codes that achieve very high coding gains while managing decoding complexity by effectively being able to decode in more than one stage. "Classical" concatenation would consist of a relatively short inner code (often a convolutional code) cascaded with a longer high-rate algebraic outer code (typically nonbinary Reed-Solomon). The composite decoder would operate in two separate, independent steps. The idea is that the Viterbi decoder typically used to decode the inner code makes errors in bursts; the nonbinary code is particularly effective in controlling these kinds of errors.

Modern concatenation makes use, additionally, of an interleaver interposed between the two encoders, and, an iterative decoding system. Even though it functions to break up error bursts between decoders, it can be argued that, in these systems, this random interleaver actually provides diversity by making the two codes essentially orthogonal [31]. The key feature in the decoding of these systems is the sharing, iteratively and cooperatively, of soft information; the soft information

passed from one decoder to the other is independent of the information it receives from that decoder. This independence of the extrinsic information is directly attributable to the action of the random interleaver. We now briefly review parallel and serial concatenation.

2.3.1 Parallel Concatenation

Figure 2.7 depicts the standard turbo encoder. It consists of two binary recursive systematic convolutional (RSC) encoders separated by a length-N fixed pseudo-random interleaver (permuter) and arranged in so-called parallel concatenation. The structure shown in Figure 2.7 can be generalized to consist of M component RSC encoders separated by $M-1$ interleavers but we only discuss the case for which $M = 2$. The arrangement facilitates puncturing to obtain any rational rate greater than or equal to $1/(M+1)$ and makes the turbo code resemble a long random block code of length N. The input sequence is thus encoded twice, being randomly rearranged before being encoded by the second encoder. The component RSC encoders need not be identical but they often are and we make this assumption in this thesis. The performance of the code is a direct function of N and this performance/interleaver size relationship has come to be known as *interleaver gain*. One of the fundamental results of information theory is that block codes become more powerful as the block size is increased, and turbo codes owe much of their popularity to the sheer ease with which large block sizes can be obtained; one simply employs a larger random interleaver. Parallel concatenation also makes it unnecessary to transmit the systematic portion of the second encoder since it is simply an interleaved version of the systematic part of the first encoder.

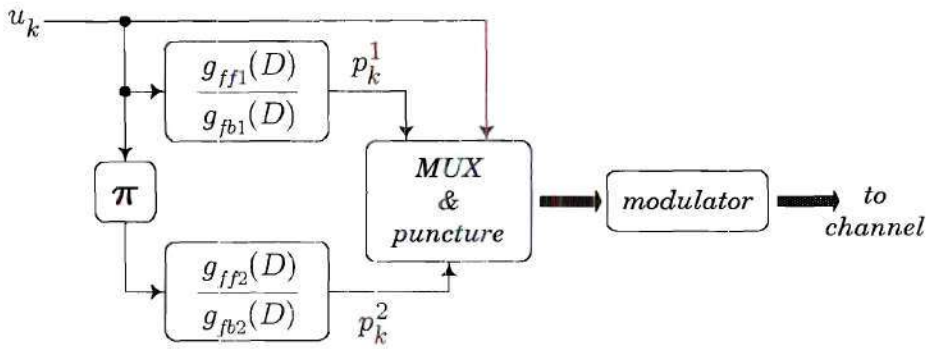


Figure 2.7. Parallel concatenated ("turbo") encoder.

The fixed interleaver size necessitates that the encoder operate on a block of data at a time. As has been demonstrated by Benedetto and Montorsi in [12] and [13] the component codes in this arrangement are required to be recursive in order to obtain the interleaver gain. For every nonrecursive convolutional encoder there is an equivalent RSC encoder that generates the *same code.* The two encoders differ, though, in the *mapping* of input sequences to codewords.[2] For example, whereas the generator matrix for a rate 1/2 nonrecursive convolutional code is of the form

$$G_{NRC}(D) = [g_{fb}(D) \quad g_{ff}(D)] \tag{2.14}$$

the equivalent RSC encoder generator matrix has the form

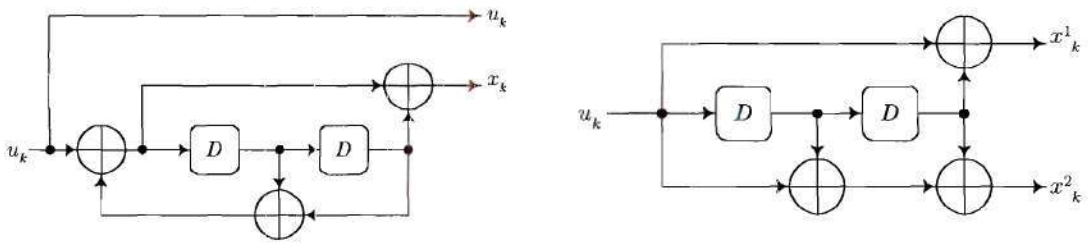
$$G_{RSC}(D) = \left[1 \quad \frac{g_{ff}(D)}{g_{fb}(D)}\right]. \tag{2.15}$$

The two encoders shown in Figure 2.8 generate the same code sequences, but each encoder maps a particular codeword $\boldsymbol{c}$ to a different input sequence. The encoders are characterized by $g_{fb}(D) = 1 + D + D^2, g_{ff}(D) = 1 + D^2$. In later chapters of this

[2]We refer to the collection of codewords (without specifying how the encoder maps them to data blocks) as the code.

thesis we will use the standard notation for designating RSCs, which for Figure 2.8(a) is $(7, 5)$. This is the octal representation $(111, 101)$ of the shift register taps.

Since all its components are linear, a turbo code is also linear and therefore many of the established methods for analyzing linear codes can be applied to turbo codes. One consequence of linearity is that $w(\boldsymbol{c}) = w(\boldsymbol{s}) + w(\boldsymbol{p}^1) + w(\boldsymbol{p}^2)$, where $w(\boldsymbol{x})$ is the Hamming weight of the sequence $\boldsymbol{x}$, and $\boldsymbol{c}$, $\boldsymbol{s}$, $\boldsymbol{p}^1$, and $\boldsymbol{p}^2$ refer to the codeword, systematic, encoder 1 parity, and encoder 2 parity bits, respectively.



(a) (b)

Figure 2.8. Two convolutional encoders that generate the same code. (a) Recursive systematic; (b) Nonrecursive nonsystematic.

The large pseudo-random interleaver, though, constitutes the chief impediment to a complete analysis of turbo codes for it endows the turbo code with time varying properties. In particular, for ordinary convolutional codes, if $\boldsymbol{x}(\boldsymbol{s})$ represents the encoder output, then

$$\boldsymbol{x}(D^i\boldsymbol{s}) = D^i\boldsymbol{x}(\boldsymbol{s}), \tag{2.16}$$

but this is not generally true for turbo codes. The function of the random interleaver is to ensure, with high probability, that $w(\boldsymbol{p}^1)$ and $w(\boldsymbol{p}^2)$ are not simultaneously small. That is, if $\boldsymbol{s}$ produces $\boldsymbol{p}^1$ with small $w(\boldsymbol{p}^1)$, then with high probability, $\pi(\boldsymbol{s})$ will not also generate $\boldsymbol{p}^2$ with small $w(\boldsymbol{p}^2)$, where $\pi(\cdot)$ denotes the interleaver function.

As is well-known, input sequences with weight at least 2 are required to produce finite-weight parity sequences from an RSC encoder. This follows from the form of the generator matrix for these codes given in (2.15). Finite weight code sequences require that $\boldsymbol{s}(D)$ be divisible by $\boldsymbol{g}_{fb}(D)$ and this requires that $w(\boldsymbol{s}) \geq 2$ for $\deg\{\boldsymbol{g}_{fb}(D)\} > 0$. Significant structure in the interleaver therefore undermines its effectiveness in 'breaking up' such input sequences that produce low-weight parity sequences from the first RSC encoder [56] .

The high coding gain achieved by turbo codes at low SNRs has been explained in terms of an interplay between the random interleaver and RSC codes termed spectral thinning by Perez *et al.* [56]. Turbo codes are said to possess a "thin" spectrum in contrast with a "dense" spectrum for convolutional codes. This is explained in terms of the weight distribution of the code and effective multiplicity. The effective multiplicity, $\tilde{N}_d$, of a codeword is defined to be the ratio of the total number of codewords with Hamming weight d to the block length of the codeword. The smaller it is the better the performance of the code because it decreases the number of error events that correspond to codewords of weight d. Perez *et al.* [56] demonstrate that for turbo codes built around RSC codes, $\tilde{N}_d \ll 1$ due to the action of the interleaver, whereas for turbo codes built around nonrecursive convolutional codes $\tilde{N}_d \propto N$. Therefore large interleavers give turbo codes very small effective multiplicity for codewords with small weight and this is known as a thin distance spectrum.

Because of their time-varying nature, an exact expression for the error probability for turbo codes is very difficult to obtain. If, however, we assume that there exists a true Maximum Likelihood (ML) decoder for turbo codes, then a union bound may be used to establish an upper bound on the turbo code bit error rate

(BER). For a block code of length N, the BER achieved by an ML decoder for BPSK in the presence of AWGN can be upper bounded by [58]

$$P_b \leq \sum_{d=d_{free}}^{N} \left[\frac{A_d \tilde{w}_d}{N}\right] Q\left(\sqrt{d \frac{2R_c E_b}{N_o}}\right), \tag{2.17}$$

where A_d is total number of codewords with Hamming weight d, and $\tilde{w}_d$ is the average weight of information sequences corresponding to codewords of weight d. Also, $Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty dy\, e^{-y^2/2}$. At moderate-to-high SNRs this sum is dominated by codewords with weight equal to the minimum free distance of the code, and thus a very good approximation of the ML asymptote is

$$P_b \approx \frac{A_{d_{free}} \tilde{w}_{d_{free}}}{N} Q\left(\sqrt{d_{free} \frac{2R_c E_b}{N_o}}\right). \tag{2.18}$$

This ML asymptote is often referred to as the *error floor* of the code. For turbo codes d_{free} is replaced with the effective free distance $d_{free\,eff}$ which is set by information sequences of weight 2 and computed as

$$d_{free\,eff} = 2 + 2z_{min} \tag{2.19}$$

where z_{min} is the weight of the lowest weight parity sequence (after puncturing) of one RSC encoder caused by input sequences of weight 2. $A_{d_{free\,eff}}$ and $\tilde{w}_{d_{free\,eff}}$ can be obtained by exhaustively searching all even weight input sequences that terminate the RSC encoder (those for which $s(D)$ divides $g_2(D)$) [12], [13]. We lastly mention that the computation of the ML asymptote above is for the average turbo code since it has been averaged over the ensemble of random interleavers. This construct, the uniform interleaver, was first introduced by Benedetto and Montorsi in [10] and [11].

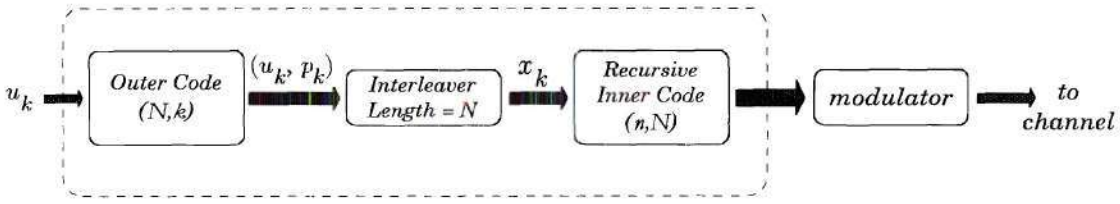


Figure 2.9. Rate k/n (n, k, N) serially concatenated code.

2.3.2 Serial Concatenation

'Modern' serial concatenation was recently introduced by Benedetto *et al.* [8]. It differs from the classical approach (Forney [25]) in the use of a random interleaver as shown in Figure 2.9. This coding structure is also decoded with an iterative decoder whose components share much in common with those used for turbo codes. Even though either or both codes can be block codes, they are usually convolutional codes since it is usually easier to obtain *a posteriori* probability (APP) information from decoders for convolutional codes. We will thus concentrate on serial concatenated convolutional codes (SCCCs). As is pointed out in [8], it is necessary for the inner code to be recursive to achieve the corresponding interleaver gain with serial concatenation. Some of the known results on serial concatenation are surprising in that the inner 'code' need not be a code at all — the use of a simple rate 1 (non redundant) recursive structure for this inner 'code' is known to show marked improvement over the performance of the outer code alone by realizing an interleaver gain [19], [54]. For application to recording channels, serial concatenation possesses the additional serendipitous advantage that a composite inner code can be formed by combining the channel PR target with its recursive precoder. This recursive precoded dicode channel (Figure 2.4(b)) is often viewed as a single entity in Viterbi detection of PR4 channels [68].

If both codes in the serial concatenated structure are linear, then so also is the overall code. The error rate performance of the structure can, again, be upper bounded by using a union bound and uniform interleaver in similar fashion to that used to analyze parallel concatenated codes. If an ML decoder is assumed, it can be shown [8] that the dominant term in the union bound ensures that

$$\lim_{N\to\infty} P_b \lessapprox \begin{cases} B_{nr} N^{\alpha_M} \exp\left(-\frac{h_M R_c E_b}{N_o}\right), & C^i \text{ non recursive} \\ B_e N^{-d_f^o/2} \exp\left(-\frac{d_f^o h_m^{(2)}}{2} R_c \frac{E_b}{N_o}\right), & C^i \text{ recursive}, d_f^o \text{ even} \\ B_o N^{-(\lfloor d_f^o+1 \rfloor/2)} \exp\left\{-\left[\frac{(d_f^o-3)h_m^{(2)}}{2} + h_m^{(3)}\right] R_c \frac{E_b}{N_o}\right\}, & C^i \text{ recursive}, d_f^o \text{ odd} \end{cases} \tag{2.20}$$

where N is the interleaver size, C^i the inner encoder, d_f^o the free distance of the outer code, $h_m^{(j)}$ the minimum weight of sequences of the inner code corresponding to weight-j input sequences, and α_M and h_M positive constants that depend on d_f^o. $B_{nr}, B_e,$ and B_o are also positive constants, and R_c is the rate of the concatenated code. The design rules distilled from the above expressions are as follows:

(1) Interleaver gain is always assured when C^i is recursive. Conversely, it is unavailable when C^i is non-recursive.

(2) The inner code must be chosen to maximize $h_m^{(2)}$ and $h_m^{(3)}$.

(3) Since the interleaver gain is equal to $N^{-d_f^o/2}$ and $N^{-(\lfloor d_f^o+1 \rfloor/2)}$ for even and odd values of d_f^o, respectively, the outer code should be chosen to have a large and possibly odd value of the free distance.

(4) Although not evident from the expression above, it is also desirable to choose C^o to be nonrecursive since these encoders generally associate fewer input bit errors with free distance error events at high SNRs.

We also note that with $\mathcal{C}^i$ recursive and large interleavers, the interleaver gain is independent of the parameters of the inner code. If, in addition, we select $\mathcal{C}^i$ with $1+D$ as a factor of the feedback polynomial, all odd-weight outer codewords are unable to terminate it and therefore do not influence the BER of the serial concatenated structure [54].

In comparison with turbo codes, serial concatenated codes do not generally exhibit the error floor that is characteristic of turbo codes. This is because turbo codes have an effective interleaver gain of $1/N$ regardless of the constituent RSC codes, whereas, as is obvious from (20), serially concatenated codes can exhibit very large interleaver gain.

2.3.3 Iterative Decoding

Due to the presence of the random interleaver, ML decoding of concatenated codes is prohibitively complex and practically unrealizable. Fortunately, both serial and parallel concatenated systems make use of iterative decoders whose performance above moderate SNRs does approach, very closely, bounds calculated assuming an ML decoder. These iterative decoders both make use of optimal cooperative APP modules which implement a modification of an algorithm originally proposed by Bahl *et al.* [6][3]. This algorithm has come to be known as the BCJR algorithm. Therefore we discuss both decoders within the same context.

A general APP module calculates the log *a posteriori* probability (LAPP) ratio of either its input or output bits (or both). For a generic bit u_k which could be either the

[3]Whereas the APP detectors (BCJR algorithm) are themselves optimal, the iterative algorithm is not.

"encoder" (i.e., any trellis) input or output, the LAPP ratio (also loosely referred to often in the literature as an LLR — log likelihood ratio) computed using the appropriate trellis takes the form

$$L(u_k) = \log\left[\frac{Pr(u_k = 1|\boldsymbol{y})}{Pr(u_k = 0|\boldsymbol{y})}\right] \tag{2.21}$$

and incorporating the encoder trellis,

$$L(u_k) = \log\left[\frac{\sum_{\mathcal{S}_1}\alpha_{k-1}(s)\cdot\gamma_k(s',s)\cdot\beta_k(s)}{\sum_{\mathcal{S}_0}\alpha_{k-1}(s)\cdot\gamma_k(s',s)\cdot\beta_k(s)}\right] \tag{2.22}$$

The disjoint sets $\mathcal{S}_1$ and $\mathcal{S}_0$ contain trellis branches on which the generic bit u_k is either a *one* or a *zero*. The terms $\alpha_k(s)$, $\beta_k(s)$, and $\gamma_k(s',s)$ are referred to as the forward recursion, backward recursion and branch probabilities, respectively. They are functions of the trellis and do not depend on whether u_k is the encoder input or output and so the computation of $L(u_k)$ when u_k is either a trellis input or output differs only in the sets $\mathcal{S}_1$ and $\mathcal{S}_0$. Turbo decoders do not require a computation of the LAPP ratio for encoder output bits but decoders for serial concatenation require LAPP ratios for both trellis input and output bits u_k for the outer encoder. For systematic encoders this LAPP ratio separates into 3 independent components—systematic, extrinsic, and *a priori*; for example APP1 may compute

$$L_1(u_k) = L_c y_k + L_{12}^{ext}(u_k) + L_{21}^{ext}(u_k) \tag{2.23}$$

using the BCJR algorithm, where *only* the extrinsic component $L_{12}^{ext}(u_k)$ is passed on to APP2 to be used as *a priori* information ($L_c y_k$ and $L_{21}^{ext}(u_k)$ are subtracted from $L_1(u_k)$). L_c is often referred to as the 'channel value' which for AWGN, is $\frac{2}{\sigma^2}$; y_k is the noisy channel output corresponding to u_k. If u_k is a trellis output bit, then y_k is a noisy modulated parity bit, whereas, if u_k is a trellis input bit, then y_k is a noisy modulated systematic bit. Additionally, as is often the case for the inner code in

serial concatenation, if the encoder is nonsystematic then there is no channel value in $L(u_k)$ [31], [60], [64]. An important issue in the BCJR algorithm is 'termination' of at least one trellis — which refers to the addition of extra bits to the encoder input to force it into the all *zeros* state at the end of an encoded block. This allows unambiguous initialization of the $\beta_k(s)$. This is especially critical in avoiding serious BER degradation when Cartesian product trellises are used to incorporate certain RLL constraints as we will see in Chapter 3. It is also necessary to know the initial state of all APP trellises to enable proper initialization of the $\alpha_k(s)$. The standard iterative decoders for turbo codes and serially concatenated codes are shown in Figs. 3.2 and 2.10, respectively. Note that in Figure 3.2 we have shown the flow of the extrinsic information $L^{ext}(u_k)$ only, whereas in Figure 2.10 we have shown the flow of the full LAPP ratios $L(\cdot)$. A good exposition of (and further details on) the decoder for serial concatenation appears in Benedetto *et al.* [9] and Ryan *et al.* [64].

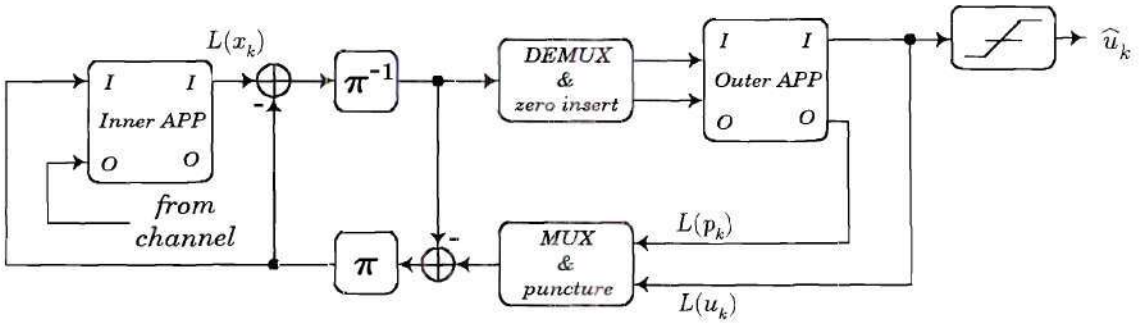


Figure 2.10. Standard iterative decoder for serial concatenation.

CHAPTER III

$(0, k)$-CONSTRAINED-INPUT TURBO CODES

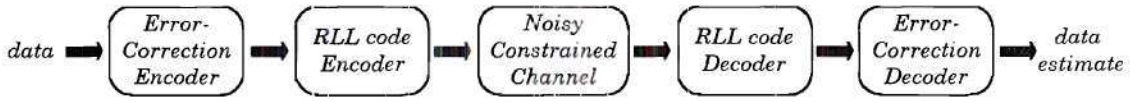
Most recent work on interleaved concatenated coding for the MRC has focussed on high-rate concatenated codes and PR signaling [22], [23], [51], [61], [64], and [68]. These papers have established the suitability of high-rate concatenated codes for PR-equalized channels. This chapter proposes a system that allows a $(0, k)$-constrained code to be used with a high-rate turbo code in such a way that it becomes possible to recover some of the rate loss of the constrained code with a higher-complexity receiver. Fan [22] and Fan & Cioffi [23] have recently shown that the modified concatenation scheme of Bliss [15] for high-rate constrained codes also readily accommodates the soft-input iterative decoders used by turbo codes. In Section 3.2 we review and analyze this configuration for comparison with the proposed system.

3.1 $(0, k)$-Constrained-Input Turbo Codes

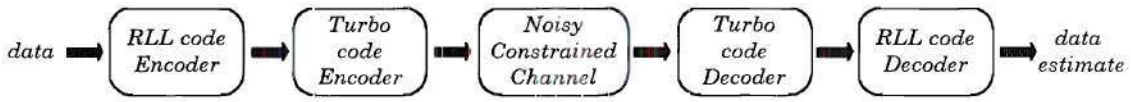
In the communication link that comprises the storage/retrieval process, the natural position of the constraint encoder is downstream of the error correction encoder, rather than *vice versa*, so that the constrained sequences can immediately be written to the channel (Figure 3.1(a)). Here we assume the error correction encoder

is a turbo code. The difficulty with this arrangement is that the decoder for the turbo code requires 'soft' inputs and it is not obvious how to produce soft channel outputs from a general $(0, k)$ code decoder.

If the turbo code rate is high, one option is to place the $(0, k)$ code encoder *upstream* of the (systematic) turbo code encoder (Figure 3.1(b)) [3].



(a) Standard Concatenation



(b) Reverse Concatenation

Figure 3.1. Standard and reverse concatenation of an ECC and constrained code.

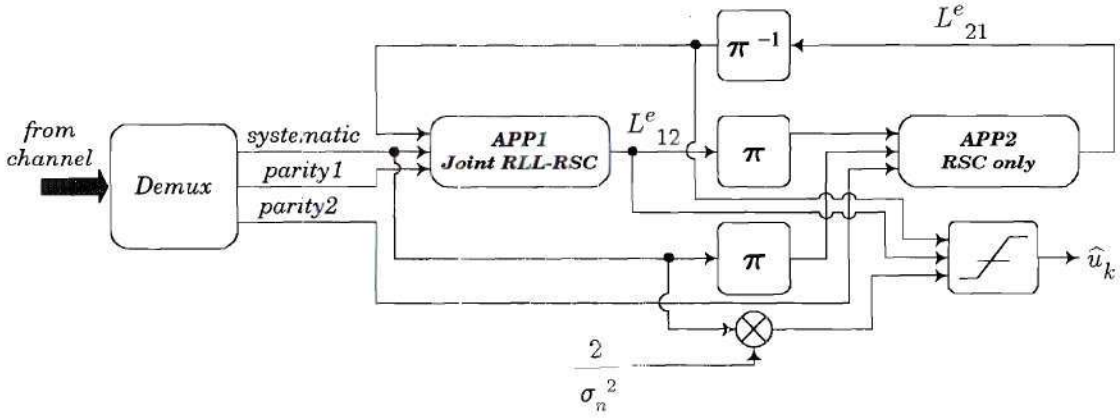
In this high-rate scenario, the injection of parity bits by the turbo code encoder into the RLL sequence will be infrequent enough to transfer, in large measure, the structure of sequences at the encoder input to those at its output. This insertion arrangement destroys a non zero d parameter (converts it to 0) but as mentioned previously, a non zero d parameter is not necessary in PRML systems. So, the turbo encoder output will also be $(0, k')$-constrained, albeit, with $k' > k$. It is easily demonstrated that the k parameter at the input to a rate $(n-1)/n$ turbo encoder is translated into

$$k' = k + 1 + \left\lfloor \frac{k}{n-1} \right\rfloor \tag{3.1}$$

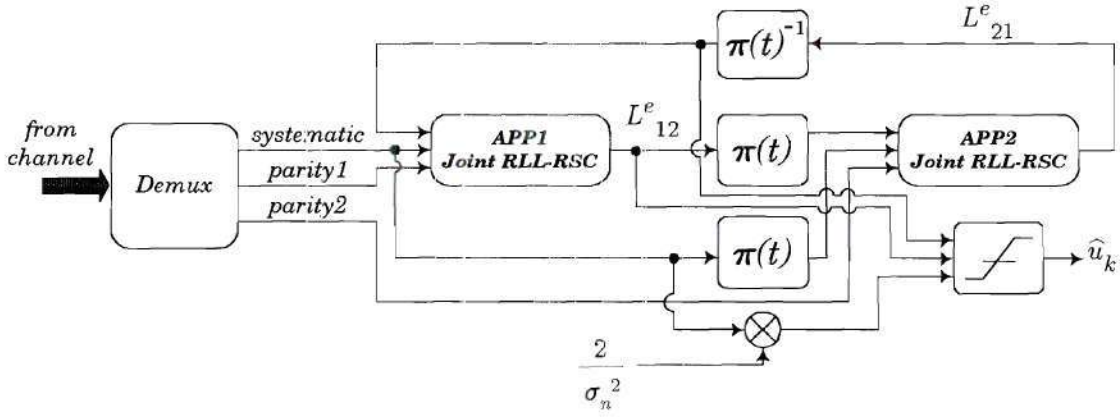
at its output. This expression is easily obtained by observing that, since the rate $(n-1)/n$ encoder inserts one parity bit for every $n-1$ systematic bits, it can insert at most $1+\lfloor \frac{k}{n-1} \rfloor$ parity bits (which could all be *zeros*) between any two *ones* that are separated by k *zeros* in the encoder input stream. On the other hand since a parity bit could be a *one*, the encoder output could have two adjacent *ones* thus destroying any $d \neq 0$ property. Therefore, for $k < n-1$, we obtain a maximum runlength value at the turbo encoder output which is only one greater than that at its input. In this configuration, we are able to integrate the $(0,k)$ constraints into the APP modules and then subsequently pass hard (binary) decisions to the $(0,k)$ code decoder without any difficulty.

It seems plausible that the underlying structure of the constraint, which can be represented by an FSTD, can be used to advantage in the turbo code decoder. In this scheme, at least one of the APP modules operates over an expanded trellis which is the Cartesian product of the constraint and turbo code constituent code trellises. In fact, a similar scheme was described by Garcia-Frias and Villasenor [29] for the case where a turbo code was driven with a hidden Markov source. We use essentially the same scheme but elaborate in greater detail on the implications of joint detection in considering $(0,k)$ codes as the 'hidden' Markov source. The motivation for our approach is not so much an attempt at joint decoding as it is one that seeks to provide to the turbo code decoder maximal *a priori* information about the input to the turbo code encoder. In what follows we consider two different decoding configurations for RSC codes under parallel concatenation only, as shown in Figure 3.2. The two configurations are considered only because the encoder interleaver is sure to cause the input to the second constituent encoder to lose its RLL properties. In configuration A (Figure 3.2(a)) only one of the constituent APP modules uses

knowledge about the $(0, k)$ constraints whereas in configuration B (Figure 3.2(b)) both APP modules make use of this additional information.



(a) Decoding configuration A



(b) Decoding configuration B

Figure 3.2. Decoding configurations for constrained-input turbo codes.

Configuration B, while perhaps not currently practical due to the use of a time-varying interleaver, provides interesting insight into the full capability of optimal decoding of constrained-input turbo codes. Next we comment briefly on the Cartesian product states.

3.1.1 $(0,k) \times$ RSC Cartesian Product Complexity

Since at least one of the APP modules within the turbo code decoder will be operating over a Cartesian product trellis we make some comments on this. To obtain the state designation of the Cartesian product states, we append the shift register contents of the turbo code constituent encoder to the binary representation of the $(0,k)$ state. The $(0,k)$ state designation is the number of successive *zeros* since the last *one*. For instance, if the $(0,k)$ state is 2 and the RSC code encoder state is 0 1, then the product state is 1 0 0 1 ($= 9_{10}$). There are $1+2k$ edges in the FSTD of the $(0,k)$ constraint and 2^{M+1} edges in the trellis of a binary-input memory M RSC trellis. The $(0,k) \times$ RSC Cartesian product can be shown to have $(1+2k)2^M$ edges even though it has $2^M(k+1)$ states (exactly 2^M of the states have only 1 outgoing/incoming edge). Thus there is an I-fold increase in the number of edges, with

$$I = \frac{(1+2k)2^M}{2^{M+1}} = \frac{1}{2} + k. \tag{3.2}$$

Since all other computations are identical, the increase in computational complexity as a ratio is also I. Figure 3.3(a) shows an example Cartesian product for the $(0,2)$ constraint driving a $(7,5)$ RSC encoder. It is important to observe that the joint trellis describes the relationship between the constrained RSC input sequences and the RSC encoder output; it is therefore be used to compute the LAPP ratios for the RSC *encoder input*—not the constraint encoder input (since we have not specified the nature of the mapping of unconstrained bits to constrained bits). This may be clarified by noting that the constraint FSTD has edges labeled with only 'outputs', as opposed to standard binary-input trellises with both 'input' and 'output' edge labels.

3.1.2 Cartesian Product Trellis Termination

A critical issue for APP1 is proper termination so that the forward and backward recursion probabilities, $\alpha_k(s)$ and $\beta_k(s)$, respectively, can be properly initialized. The necessity of this has long been established for turbo decoding [57]. Correct termination of sequences for the joint trellis requires that the termination (or 'tail') bits also be subject to the constraint. This allows for the correct initialization of the backward recursion probabilities $\beta_k(s)$ according to,

$$\beta_{N_{term}}^{(term)}(s) = \begin{cases} 1, & s = 0 \\ 0, & s \neq 0 \end{cases} \tag{3.3}$$

for terminated trellises, where the superscript $(term)$ denotes "terminated trellis" and N_{term} is the total terminated block length (N + tail bits). This results in the requirement of varying tail lengths for encoded blocks as we now demonstrate by example. Figure 3.3(b) is the termination tree which shows the state transitions required to terminate the trellis of Figure 3.3(a). If the block $\boldsymbol{y}_1 = \{1, 0, 1, 1, 1, 1, 0\}$ is encoded beginning in state 0, the final state is 5 after completion. $\boldsymbol{y}_1$ requires a termination tail of length 1 ($= \{1\}$). On the other hand, the terminal state would be 7 after encoding $\boldsymbol{y}_2 = \{0, 1, 1, 1, 0, 1, 0\}$ which would require a tail of length 3 ($= \{1, 0, 1\}$). It is clear, then, that we cannot simply always pad the tail bits with an arbitrary number of *zeros* or *ones* (or any combination of them) to obtain fixed turbo code block sizes as is feasible with an RSC trellis. However, since the maximum number of tail bits that can be required for termination is known in advance, we can use this number to set the 'mother' interleaver size and then 'puncture' the interleaver as necessary on a block by block basis according to a prearranged rule. For instance, the mother length-10 interleaver might be $\pi = \{5, 1, 8, 4, 2, 9, 7, 10, 3, 6\}$, accommodating the largest expected tail length of 3 for our example trellis (implying $N = 7$). If no tail is required for a particular block we use $\pi''' = \{5, 1, 4, 2, 7, 3, 6\}$.

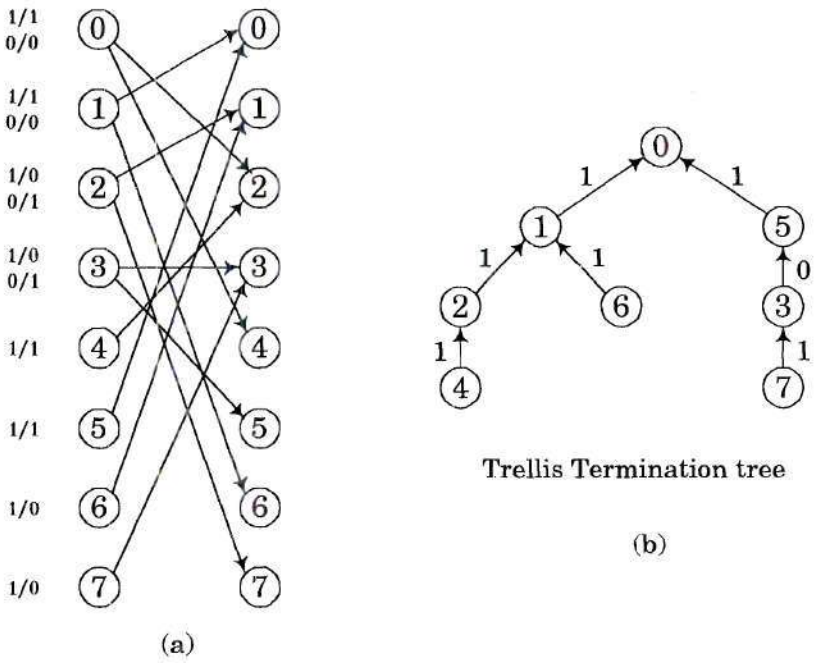


Figure 3.3. (a) Cartesian product trellis of $(0, 2)$ constraint driving the $(7, 5)$ RSC encoder; (b) trellis termination tree.

If just one tail bit is required we use $\pi'' = \{5, 1, 8, 4, 2, 7, 3, 6\}$ and so on. If a particular distance is required for the interleaver, one ensures that it is available when the interleaver is maximally punctured since each punctured position reduces the distance by one.

3.1.3 Cartesian Product Trellis Initialization

The $(0, k)$ state at the end of a sequence can be uniquely determined by simply counting, from the end of the sequence, the number of *zero*s before encountering the first *one*. So both $\boldsymbol{y}_1$and $\boldsymbol{y}_2$ above terminate in $(0, k)$ state 1 whereas the sequence $\{1, 1, 1, 0, 1\}$ terminates in $(0, k)$ state 0. Knowledge of the terminal state and the sequence, however, does not permit unambiguous identification of the initial $(0, k)$ state (not the case for a general convolutional code trellis). We clarify this by aug-

menting our description of $(0,k)$ constraints. These constraints are also embodied in *phrases* which we discussed in Section 2.2.2. As an example, we can parse the sequence $\boldsymbol{y}_2$ as $\{\underline{01},\underline{1},\underline{1}\ \underline{01},\underline{0}\}$, with the last *phrase* incomplete. In this simple 2-state $(0,1)$ case, it is clear that the sequence could not have originated from state 1. More generally, if $\boldsymbol{y}_2$ were the output from a $(0,k)$ graph, the only beginning state which is not possible is state k. The initial state S_I is thus a random variable whose distribution, conditioned on the event that the initial *phrase* length $L_I = l$, can be taken as uniform and expressed as

$$p_{S_I|L_I}(l) = \frac{1}{k-l+2},\ 1 \le l \le k+1. \tag{3.4}$$

For given k, if the initial *phrase* length $l = k+1$ there is no uncertainty in the initial state. Conversely, if the *phrase* length is $l = 1$ there is maximal uncertainty. For a sufficiently long binary $(0,k)$ sequence derived from a rate R code, the *phrase* length distribution is

given by

$$p_{L_I}(l) = \frac{2^{-lR}}{\sum_{n=1}^{k+1} 2^{-nR}} \tag{3.5}$$

Therefore long initial *phrases*, which from (3.4) reduce the ambiguity in the initial state, are the least likely to occur. And the situation is exacerbated by raising the code rate.

This ambiguity moves into the product trellis and makes correct initialization of the product trellis APP module unlikely. If the constrained code encoder is implemented as a finite-state type (designed, for instance, by the state splitting algorithm—see Section 2.2.1), it is possible to ensure that all (turbo encoder) blocks are begun from $(0,k)$ state 0 thus removing the ambiguity. If on the other hand the

encoder is implemented with an enumerative scheme (Section 2.2.2) or other *ad hoc* means, the ambiguity remains. One solution to this is to communicate to the decoder, for each block, the set of possible initial states based on the initial *phrase*. Taking the distribution of (initial) *phrase* lengths to be as given in (3.5), this communication will require at least

$$H(p_{L_I}(l)) = \frac{R\sum_{l=1}^{k+1} 2^{-lR} l}{\sum_{n=1}^{k+1} 2^{-nR}} \text{bits} \tag{3.6}$$

where $H(\,\cdot\,)$ denotes the entropy function. For $k < 4$ this requirement is about 2 bits depending on the actual code rate. Initialization of the forward recursion probabilities could then be set uniformly over this state subset by,

$$\alpha_0(s) = \begin{cases} \frac{1}{|S_I(t)|}, & s \in S_I(t) \\ 0, & s \notin S_I(t) \end{cases} \tag{3.7}$$

and the time dependence of the set of possible initial states S_I is explicitly shown. Notwithstanding these comments, we have observed that the turbo decoder performance is not very sensitive to the actual initial $(0, k)$ state when the usual initialization

$$\alpha_0(s) = \begin{cases} 1, & s = 0 \\ 0, & s \neq 0 \end{cases} \tag{3.8}$$

is used. This is presumably because the initial state ambiguity is inherent in the $(0, k)$ constraints.

3.1.4 Simulation Results and Discussion

Using BPSK modulation on an AWGN channel, we have simulated the performance of joint decoding using both configurations in Figure 3.2. We have done this for

several values of k using several turbo code encoders. In order to investigate variations of $(0, k)$ codes at arbitrary rates below $C_{(0,k)}$ we simply drive the turbo encoder with outputs from a maxentropic constraint graph whose rate we can arbitrarily set to reflect varying efficiencies. By maxentropic constraint graph, we mean one whose state transition probabilities have been computed to maximize the entropy of the graph. Justesen [35] has shown that these maxentropic transition probabilities $\{p_{ij}\}$ from state i to state j are given by

$$p_{ij} = \frac{v_j}{\lambda_{max} v_i} \qquad i, j \in \{0, 1, ..., k\} \tag{3.9}$$

where v_i and v_j are components of the right eigenvector $\boldsymbol{v}$ of $\boldsymbol{T}$ corresponding to the eigenvalue λ_{max}. The resulting entropy of this maxentropic constrained graph is then equal to $C_{(0,k)}$. Such a source allows us to work with the highest possible rates available from the constraint. Therefore, in Figure 3.1(b) the input to the turbo encoder is this source and all error rates are obtained for the output of this source. An actual constrained code may possess the additional property of nonuniform prior probabilities that we have seen to sometimes quicken convergence of the iterative algorithm with normal turbo decoding. Use of the maxentropic constraint graph approach also has practical implications. In fact, use of the constraint graph instead of the code itself in decoding, is not a new idea; it was originally proposed by Karabed & Siegel for MSN codes [37].

Naïve Decoder (Reference): We refer the performance of configurations A & B in Figure 3.2 to that of a "naïve decoder" both of whose APP modules are based on the RSC trellises only. The naive decoder thus passes hard naïve decisions to the constraint decoder. Simulation of this system using the naïve decoder and configurations A & B respectively, reveal several interesting results. First, with the

naïve decoder, the behavior of the BER versus SNR (E_b/N_o) curve can be completely accounted for by the nominal rate loss of the constrained code. This is due to the fact that $(0, k)$ codes are not endowed with any distance properties (their minimum distance is 1). For example, Figure 3.4 curve B, which is for the naïve decoder using a rate 0.84 $(0, 2)$ code, shows a rate loss of about 0.8 dB. On an AWGN channel with BPSK signaling, a code rate of 0.84 without attendant increase in minimum distance carries a rate loss penalty of $10\log(1/0.84) \approx 0.8$ dB also.

Configuration A: Decoding using configuration A is able to recover about 0.4 dB of the nominal rate loss of a 95% efficient, rate 0.84 $(0, 2)$ code, as shown in Figure 3.4 curve C. So we are able to trade off higher complexity in APP1 to recover some rate loss. This configuration may be found useful for modest rate loss recovery for $k < 4$. As k increases to 4 and beyond, the complexity/rate-loss-recovery tradeoff displays greatly diminishing returns and is perhaps, not worth the effort. This is consistent with the observation that, as $k \to \infty$, we approach an unconstrained sequence and the Cartesian product should do no better than the RSC trellis alone.

Simulation of this configuration for different RSC polynomials and turbo code rates indicates that the rate loss recovery depends only on k and not on the RSC polynomials or turbo code rate. Table 3.1 shows the nominal rate loss and the actual rate loss sustained by Configuration A for some values of k . For $k = 3$ and $k = 4$ we observe that net coding gain is actually possible if we can employ a very high-rate ($\gtrsim 97.5\%$ efficient) constrained code. Also evident is the fact that configuration A performs worse than the less complex naïve decoder for low efficiencies.

Configuration B: We employ a data-dependent time-varying interleaver to make Configuration B possible. For each block, the encoder selects a permutation that preserves the constraint.

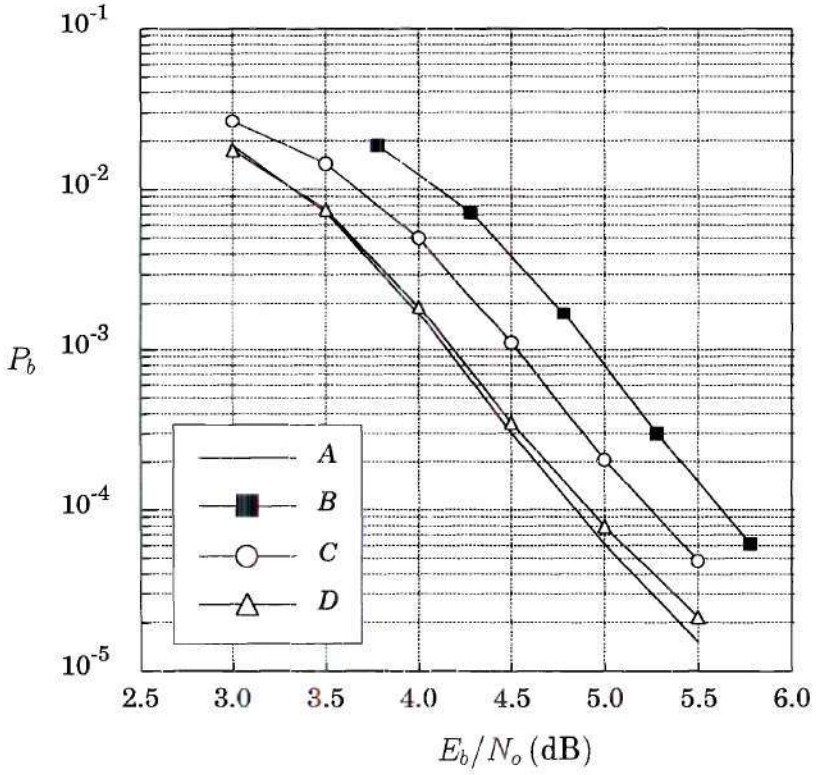


Figure 3.4. Outer RLL constrained code/inner turbo code (Figure 3.1(b)) under several types of decoding. 5 iterations. RLL code is a rate .84 95% efficient $(0, 2)$ code; turbo code is rate 8/9 (7,5,1000). *B*—naïve decoder; *C*—decoding configuration *A*; *D*—decoding configuration *B*; *A*—turbo code only (for reference).

Table 3.1. Configuration A vs. nominal (naïve decoder) rate loss (dB) for some values of k at different efficiencies. Bold entries indicate net coding gain. For entries x/y, x is the rate loss in configuration A, y is the nominal rate loss, and $y - x$ is therefore the rate loss recovery due to Configuration A.

$k\backslash\eta$	85%	90%	95%	97.5%	100%
2	1.4/1.3	0.8/1.0	0.4/0.8	0.2/0.7	0.0/0.6
3	1.0/0.9	0.5/0.7	0.1/0.5	**−0.2**/0.3	**−0.4**/0.2
4	1.2/0.8	0.8/0.6	0.2/0.3	0.0/0.2	**−0.2**/0.1

This permutation has to be communicated to the decoder on a block by block basis and so this configuration is not practical—nevertheless, the results are interesting and instructive. We have investigated two kinds of constraint-preserving interleavers for this configuration. The first, in which the actual permutation is derived from *phrase*-level interleaving, is generally implemented as follows. The sequence is first parsed into *phrases*. The *phrases* are then interleaved (which does not break up any run of *zeros*) and then, finally, the individual bit indices are carried along in exactly the same way. For example, $\boldsymbol{y}_2 = \{\underline{01}, \underline{1}, \underline{1}, \underline{01}, \underline{0}\}$ has 5 *phrases*. We may therefore interleave the set of *phrases* $\{1, 2, 3, 4, 5\}$ to yield, $\boldsymbol{\pi}^{(ph)} = \{4, 1, 3, 5, 2\}$. The final permutation is obtained by arranging the indices of the bits in the *phrases* of $\boldsymbol{y}_2$ in the same order as $\boldsymbol{\pi}^{(ph)}$, keeping all indices in a *phrase* together. This gives $\boldsymbol{\pi} = \{5, 6, 1, 2, 4, 7, 3\}$. Using (3.5), and noting that the expected *phrase* length is also the expected value of the number of bits/*phrase*, there are, on average,

$$f_b = \frac{\sum_{l=1}^{k+1} 2^{-lR}}{\sum_{l=1}^{k+1} 2^{-lR} l} \text{ phrases/bit} \tag{3.10}$$

giving an average effective interleaver size of $f_b N_{term}$. For example, a rate $8/9$ $(0,3)$ code would have about 0.56 *phrases*/bit, making a block size of 1000 behave more like one of size 560. We have nevertheless found this interleaver to perform much more poorly than we expected, due perhaps, to the fact that the *phrases* of length greater than 1 force too many successive indices to mapped to exactly the same order. We next describe a much better interleaver. We first find the indices of the *zeros* and *ones*, respectively, in the sequence, say $\boldsymbol{z}$ and $\boldsymbol{o}$. We also form the vector $\boldsymbol{r}$ of runlengths of *zeros* from the constrained sequence (the components r_i of $\boldsymbol{r}$ are drawn from $\{0, 1, ..., k\}$). We next randomize $\boldsymbol{z}$, $\boldsymbol{o}$, and $\boldsymbol{r}$ to obtain $\boldsymbol{z}'$, $\boldsymbol{o}'$, and $\boldsymbol{r}'$. Finally, we sequentially insert the $\{o'_i\}$ into the $\{z_i\}$ spacing o'_i and o'_{i+1} by r_j. For instance, if $\boldsymbol{y} = \{1,0,0,1,0,1,1,0,0,1,0,0,0\}$ from the $(0,3)$ constraint, then perhaps $\boldsymbol{z}' = \{3,13,5,2,9,12,8,11\}$, $\boldsymbol{o}' = \{10,6,7,4,1\}$, and $\boldsymbol{r}' = \{3,0,2,2,0,1\}$. We then obtain $\boldsymbol{\pi} = \{3,13,5,10,6,2,9,7,12,8,4,1,11\}$, which applied to $\boldsymbol{y}$ gives $\pi(\boldsymbol{y}) = \{0,0,0,1,1,$ $0,0,1,0,0,1,1,0\}$. This procedure gives very good results, suggesting that the interleavers derived this way have no perceivable structure.

Essentially then, the results of this simulation reflect the average performance of the configuration, much like the theoretical Uniform Interleaver introduced by Benedetto & Montorsi [11]. While not practical, it shows the interesting result that when we are able to completely exploit the constraint structure in both APP1 and APP2, each contributes an equal amount of rate loss recovery. Curve D in Figure 3.4 shows that practically all of the rate loss is recovered with this decoding arrangement.

This result provides motivation for investigating whether net coding gain is possible with respect to the turbo code alone. We describe one mechanism by which this can happen. If we forbid the appearance of certain strings within the $(0,k)$ sequence, we further reduce its capacity. But since we have not altered k this altered

constraint will, with Configuration A, perform just as well as the pure $(0,k)$ constraint. By this we mean that the performance curves for both of these cases will almost coincide. If we take the lower rate of the modified constraint into account, we then see that it provides greater rate loss recovery than the pure $(0,k)$ constraint. If we then use this modified constraint in Configuration B we can get a significant coding gain above the performance of the turbo code alone. We give an example. Borrowing some notation from Karabed & Siegel [38], Let X_{101} refer to the constraint which forbids the string 101 from occurring in a sequence. If we additionally impose the $(0,4)$ constraint, the FSTD for this combined $(0,4)X_{101}$ constraint is given in Figure 3.5.

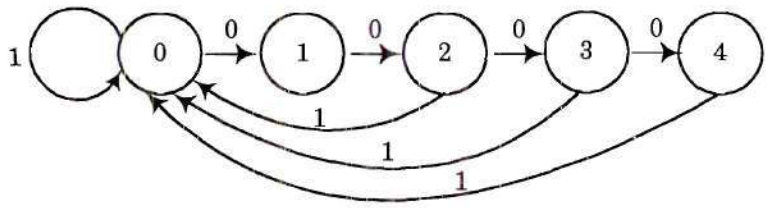


Figure 3.5. FSTD for the $(0,4)X_{101}$ constraint.

This constraint has capacity $C_{(0,4)X_{101}} = 0.7528$; the pure $(0,4)$ constraint has capacity $C_{(0,4)} = 0.9752$. This implies that the $(0,4)X_{101}$ constraint will show approximately an additional $10\log(0.9752/0.7528) = 1.1$ dB per APP module over the $(0,4)$ case (in the sense explained above). Of all the $X_{10(j)1}(0,k)$ constraints, $j = 1$ results in the lowest capacity since it forbids the shortest *phrase* without undoing the constraint. It therefore gives the highest rate loss recovery. If we use this constraint in Configuration B we see a substantial coding gain as demonstrated in Figure 3.6. We reiterate that this coding gain is independent of the code polynomials used in the turbo encoder. This result suggests that $(0,k)$ constraints, which have no distance,

can be capable of coding gain.

3.1.5 Constrained-Input Turbo Codes and Error Events

We can also contemplate using constrained codes to eliminate certain error events for some turbo code RSC polynomials in this configuration. But, by definition, APP detectors, even those based on a Cartesian product or constraint trellis, can decode to unconstrained sequences—those that violate the constraint. This is due to the fact that MAP decoding (which the APP algorithm implements) is not ML and so the APP algorithm does not always decode to a trellis input sequence that corresponds to a valid trellis output sequence. Therefore APP-based iterative decoding cannot take advantage of the ability of properly constrained sequences to eliminate some minimum distance error events for some turbo code encoder feedback polynomials. The required constraints are so severe, however, that the corresponding rates are very low and of questionable utility. For an ML decoder to eliminate a particular error event when a trellis is driven by a constraint, it is necessary for the constraint to be such that no two constrained sequences of any fixed length can differ by that error event. Consider, for instance, error events with D transform of the form

$$e(D) = 1 + D^{d'}. \tag{3.11}$$

Let us consider a general (d, k) constraint with which we wish to eliminate $e(D)$ in the encoder input sequence with an ML decoder. Let

$$x_1(D) = D^i\left(1 + D^{d+1} + D^{d+k+2}\right), i \in \mathbb{Z}^+$$

be a (d, k)-constrained encoder input sequence. Observe that if $x_2(D)$ is such that

$$x_2(D) = D^i e(D) \oplus x_1(D)$$

then it can be verified that $x_2(D)$ cannot also be (d, k)-constrained unless

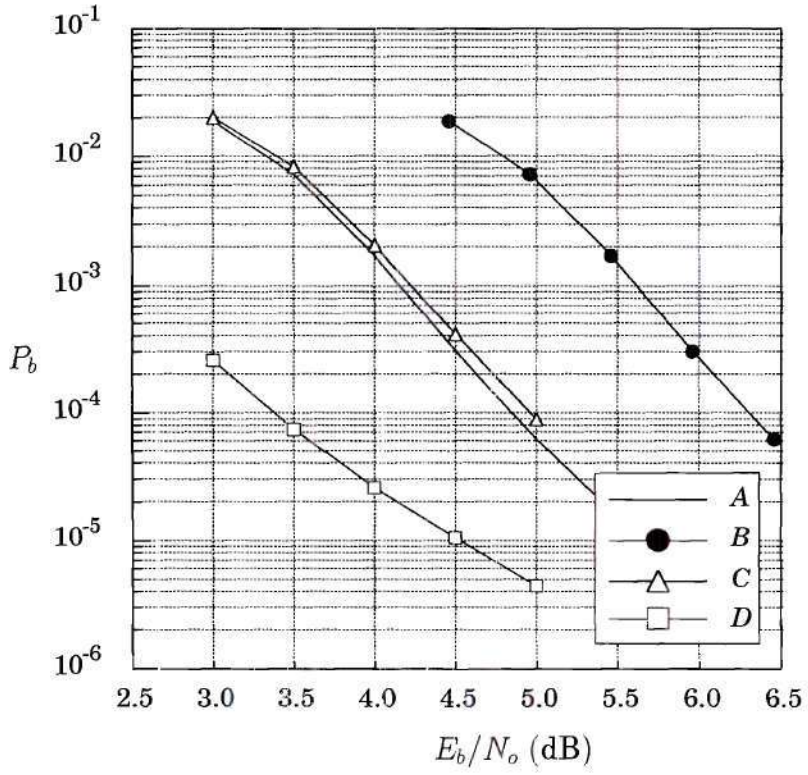


Figure 3.6. Outer RLL constrained code/inner turbo code (Figure 3.1(b)) under several types of decoding. 5 iterations. RLL code is a rate .72, 95% efficient $(0,4)X_{101}$ source; turbo code is rate 8/9 (7,5,1000). *B*—naïve decoder; *C*—decoding configuration *A*; *D*—decoding configuration *B*; *A*—turbo code only (for reference).

$$d' - 1 \leq d, \Rightarrow d_{min} = d'; \text{ and,}$$
$$d + d' > k, \Rightarrow k_{max} = d + d' - 1, \text{ or } k_{max} = 2d' - 1.$$

So a $(d', 2d' - 1)$ constraint is required to eliminate error events of the form $1 + D^{d'}$. Consider, for example, $g_{fb}(D) = 1 + D^2$ for the $(7, 5)$ RSC encoder. The predominant error event is also $e(D) = 1 + D^2$. In this case a $(2, 3)$ constrained code would eliminate this error event in the presence of an ML decoder but the capacity of this constraint is only 0.2877. We however make no claim that constraints of the form $(d', 2d' - 1)$ are the only constraints, or even that they possess the greatest capacity of all constraints, that can eliminate input error events of the form $1 + D^{d'}$. This general philosophy has been effectively applied to coding for PR systems in [39].

3.2 Comparison to Modified Concatenation

In [22] and [23] it is demonstrated that the modified concatenation scheme originally introduced by Bliss [15] can readily accommodate a soft-input decoder for a systematic ECC code and a constrained code (Figure 3.7). The scheme makes use of two distinct modulation codes for the same constraint—a long high-rate code C1 of arbitrary design, and a specialized shorter low-rate code C2 whose demodulator is able to produce soft information for the unconstrained modulator input w_k from soft information corresponding to the constrained bits z_k. This arrangement allows the soft-input systematic ECC decoder access to soft information for both the systematic and parity bits x_k and y_k, respectively. If the ECC decoder has high rate then the overall system will operate at an equivalent constrained rate only slightly lower than that of C1. Fan [22] shows several decoders C2 (including some systematic modulation codes) that can output soft information.

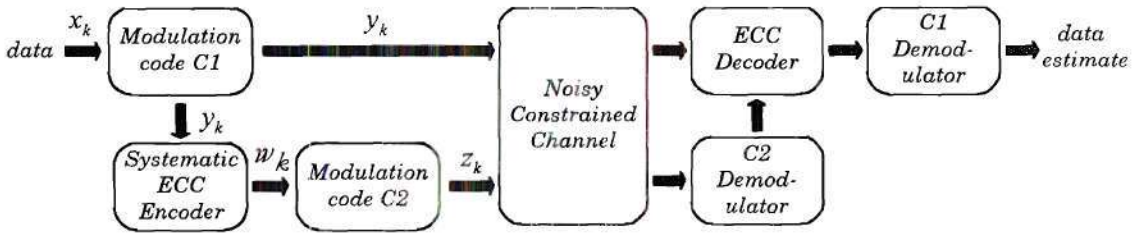


Figure 3.7. Modified concatenation.

He also shows that for a low density parity check (LDPC) code an additional 'constraint decoder' (not shown in Figure 3.7), which is the BCJR algorithm applied to the trellis of the (d, k) constraint, can provide additional gain. We will analyze the capability of this scheme, vis-à-vis overall rate loss, for the $(0, k)$-constrained channel when C2 is a systematic modulation code. We will compare the results to those for the proposed constrained-input system.

Let C1 have rate K_1/N_1, C2 have rate K_2/N_2, and the systematic ECC have rate $(n-1)/n$. We assume m_o user bits $\boldsymbol{x}$ are input into C1 producing $\frac{N_1}{K_1}m_o$ constrained channel bits $\boldsymbol{y}$. The systematic ECC then generates p' parity bits $\boldsymbol{w}$ which are then encoded by C2 to give $\frac{N_2}{K_2}p'$ bits $\boldsymbol{z}$.

So as an intermediate result, assuming m_o is large[4], we have

$$p' = \frac{N_1}{K_1} \cdot \frac{m_o}{n-1}.$$

Let C1 have efficiency η and let us denote the capacity of the (d, k) constraint by $C_{(d,k)}$. Then since we have m_o user bits into the system and a total of

$$\frac{N_1}{K_1}m_o + \frac{N_2}{K_2}p'$$

[4] This just allows us to write $\left\lfloor \frac{N_1}{K_1} \cdot \frac{m_o}{n-1} \right\rfloor = \frac{N_1}{K_1} \cdot \frac{m_o}{n-1}$.

$$= m_o\left[\frac{1}{\eta \cdot C_{(d,k)}} + \left(\frac{N_2}{K_2}\right)\left(\frac{1}{\eta \cdot C_{(d,k)}}\right)\left(\frac{1}{n-1}\right)\right]$$

output bits, the overall rate expansion R'_{MC} accounting for both the constraint and the ECC is

$$R'_{MC} = \frac{1}{\eta \cdot C_{(d,k)}} + \left(\frac{N_2}{K_2}\right)\left(\frac{1}{\eta \cdot C_{(d,k)}}\right)\left(\frac{1}{n-1}\right). \tag{12}$$

But if we considered only a rate $(n-1)/n$ ECC, m_o user bits would result in $m_o n/(n-1)$ coded bits. So, with modified concatenation, the equivalent rate expansion due to just the constraint is

$$R_{MC} = \frac{n-1}{n}\left[\frac{1}{\eta \cdot C_{(d,k)}} + \left(\frac{N_2}{K_2}\right)\left(\frac{1}{\eta \cdot C_{(d,k)}}\right)\left(\frac{1}{n-1}\right)\right]. \tag{3.13}$$

For C2 a systematic $(0, k)$ block code, $N_2 = k+1$ and $K_2 = k$. Assuming a rate $8/9$ ECC, we have tabulated, in Table 3.2, the rate loss that would be suffered by this system for several values of the runlength k for some high values of η.

A direct comparison between the entries in Tables 3.1 and 3.2 is not exactly fair since the decoder for the system in Figure 3.2(a) is more complex. But Fan also shows in [23] that the addition of the constraint decoder (for the $(0, 3)$ case for example—which increases the complexity of the system towards that of Figure 3.2(a)) can supply an additional 0.3 dB for a LDPC ECC. This would result in a best case overall rate loss for modified concatenation (for the $(0, 3)$ case) of 0.1 dB — compared with a maximum possible coding gain of 0.4 dB for the decoder in Figure 3.2(a). Therefore this additional complexity rewards us with significantly better performance (0.5 dB) when a very high-efficiency constrained code is available. On the other hand, for $\eta = 95\%$ the difference between the two cases is reduced to 0.2 dB.

Note, though, that since the input to the systematic ECC encoder is (d, k)-constrained, a Cartesian product trellis could also be used to improve the performance of the ECC decoder and reclaim some rate loss as we have described.

Table 3.2. Overall rate loss of modified concatenation for some constraints. Assumption is that C2 is a systematic code. $(1, 3)$ entries have assumed the rate $1/2$ MFM code (Section 2.2).

$(d, k)\backslash\eta$	95%	97.5%	99%	100%
$(0, 2)$	1.0	0.9	0.8	0.8
$(0, 3)$	0.6	0.5	0.4	0.4
$(0, 4)$	0.5	0.3	0.3	0.2
$(0, 5)$	0.4	0.3	0.2	0.1
$(0, 6)$	0.3	0.2	0.1	0.1
$(1, 3)$	3.3	3.2	3.1	3.0

CHAPTER IV

PERFORMANCE OF HIGH-RATE $(0, k)$ BLOCK CODES

In this chapter we turn our attention to a consideration of $(0, k)$-constrained block codes with rate $(n-1)/n$. We show that good $(0, k)$-constrained block codes of rate $(n-1)/n$ and with $d_{min} = 1$ exist. Furthermore, during the analysis, the construction of several families of such codes will be shown. By 'good' we mean that, despite the fact that the codes have $d_{min} = 1$, the combined code and encoder, with an ML decoder, can still be shown to display substantial rate loss recovery. This is tantamount to the error rate at the ML decoder output being superior to that at its input. The results from this analysis and code construction will be used in the next chapter when we address a system where a turbo code is cascaded with a $(0, k)$-constrained block code, as in the standard recording configuration of Figure 3.1(a), [5].

The motivation for this investigation is the need to know what performance is possible from $(0, k)$-constrained codes implemented as simple block codes when used in the system of Figure 3.1(a). As pointed out earlier, it is generally not known how to obtain soft information from decoders for most high-rate constrained codes because their encoders are complex and therefore obscure the relationship between encoder input and output bits. We are therefore limiting the implementation of the $(0, k)$-constrained code to a block code since the data word-to-codeword mapping of

such an encoder is straightforward. The apparent penalty for using a block code is that we do not achieve very tight (small k) $(0,k)$ constraints—or equivalently, the code efficiencies are low. This outer ECC/inner constrained code configuration requires that the decoder for the block $(0,k)$-constrained code be able to deliver soft information to the soft-input decoder for the ECC. By analyzing some properties of $(0,k)$-constrained rate $(n-1)/n$ block codes in isolation we can make some critical observations of how such codes would affect the error-rate performance of the overall system of Figure 3.1(a). We begin by motivating the analysis with two examples of the performance of $(0,k)$-constrained rate $(n-1)/n$ block codes with an ML decoder.

4.1 Two Motivating Examples

In the next section we derive an expression for a union bound to the input bit error rate for block codes. Here we provide two examples which provide the motivation for that discussion. We focus on an AWGN channel and some $d_{min} = 1$ block $(0,k)$ codes.

4.1.1 A Simple Rate $2/3$ $(0,3)$ $d_{min} = 1$ Code

As a first motivating example, consider the following simple nonlinear rate $2/3$ $(0,3)$ block code with $d_{min} = 1$;

Table 4.1. Rate $2/3$ $(0,3)$ block code.

$d_0 d_1$	$c_0 c_1 c_2$
00	011
01	111
10	010
11	100

For the simple system of Figure 4.1, the effective rate loss of the code is only about 1.2 dB (as shown in Figure 4.2) compared with a nominal $10\log_{10}(3/2) = 1.8$ dB; supplying a modest 0.6 dB of gain from the code/encoder structure. However, it can also be seen that the rate loss increases with increasing E_b/N_o. The error rate for the BPSK curve is just,

$$P_b = Q\left(\sqrt{2 \cdot \frac{E_b}{N_o}}\right). \tag{4.1}$$

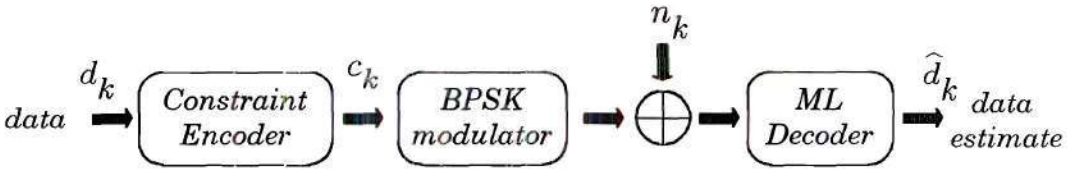


Figure 4.1. ML decoder test configuration.

So even though the code has $d_{min} = 1$ it is capable of significant rate loss recovery due to some properties of the code/encoder structure which will be highlighted in the next section.

4.1.2 Two Rate 4/5 $(0, k)$ $d_{min} = 1$ Codes

This example shows how two different codes with the same rate can differ markedly when used in the scheme of Figure 4.1. The two codes are listed together with specific encoders in Table 4.2; both have nominal rate loss of $10\log_{10}(5/4) = 1$ dB. Figure 4.3 shows how these codes perform in the configuration of Figure 4.1. Note that in Figure 4.3 $C_2(d)$ recovers essentially all of its rate loss whereas $C_1(d)$ reflects a "coding loss" (cannot recover any portion of its rate loss).

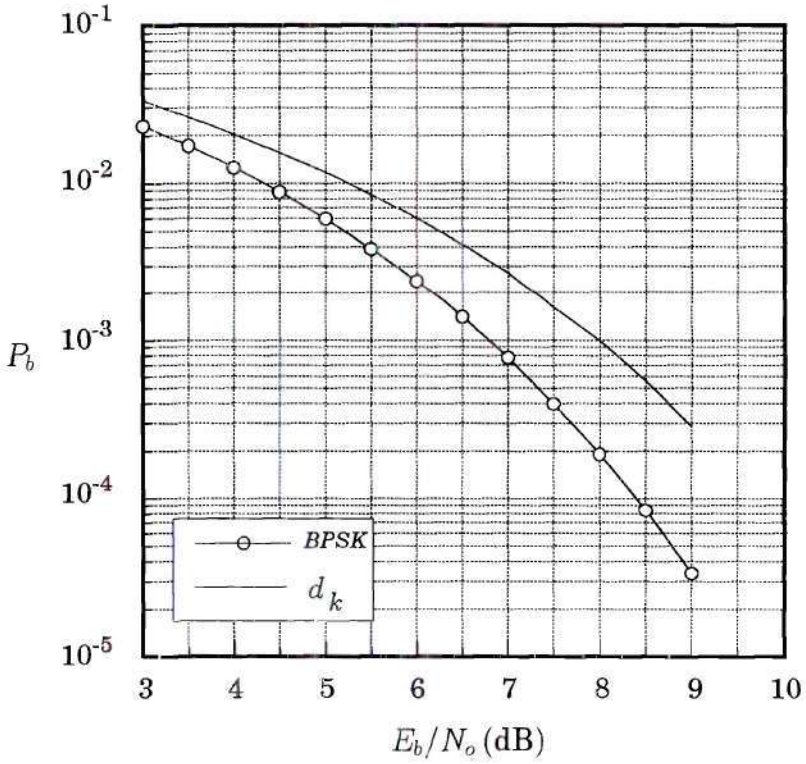


Figure 4.2. BER for code in Table 4.1: error rate for d_k in system of Figure 4.1 vs. BPSK only.

Table 4.2. Two rate 4/5 $d_{min} = 1$ constraint codes. $\mathcal{C}_1(\boldsymbol{d})$ is the $(0, 8)$ MTR code of Moon & Brickner [53] with a particular encoder; $\mathcal{C}_2(\boldsymbol{d})$ is a $(0, 6)$ code that has been specially designed.

$\boldsymbol{d}$	$\mathcal{C}_1(\boldsymbol{d})$	$\mathcal{C}_2(\boldsymbol{d})$	$\boldsymbol{d}$	$\mathcal{C}_1(\boldsymbol{d})$	$\mathcal{C}_2(\boldsymbol{d})$
0000	00001	00011	1000	01100	10011
0001	00010	00010	1001	01101	10001
0010	00100	00111	1010	10000	10110
0011	00101	00100	1011	10001	10101
0100	00110	01011	1100	10010	11010
0101	01000	01000	1101	10100	11001
0110	01001	01101	1110	10101	11100
0111	01010	01110	1111	10110	11111

Both of these examples are interesting in that they show that nonlinear block $(0, k)$ codes with $d_{min} = 1$ can exhibit performance which varies widely. Therefore clearly, the minimum distance alone of the code does not adequately characterize the error-rate performance of the code. Obviously, if the codes possess $d_{min} \geq 2$, we would expect to see the same (partial at least) rate loss recovery effect shown by these two examples. We next provide an explanation of the behavior shown in the two motivating examples by highlighting certain parameters of general block codes using a modified expression for the union bound for the BER achieved by an ML decoder.

Note that the behavior exhibited by the codes used in the two motivating examples would be essentially unchanged if the ML decoder were replaced by a MAP decoder—because the MAP decoder for a general block code performs similarly to an ML decoder above moderate SNRs.

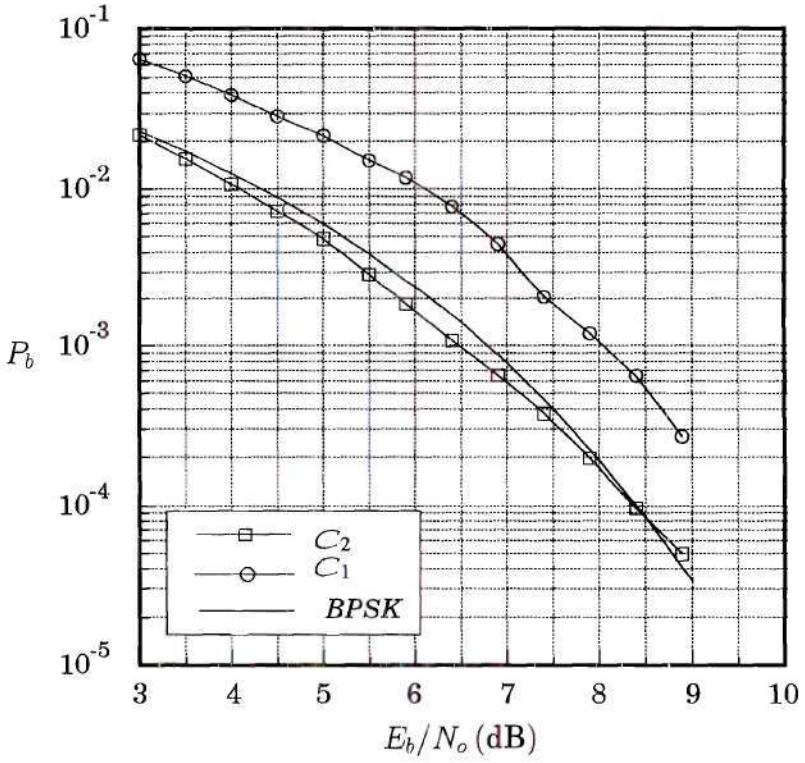


Figure 4.3. BER for codes $\mathcal{C}_1(\boldsymbol{d})$ and $\mathcal{C}_2(\boldsymbol{d})$ in Table 4.2: d_k in system of Figure 4.1 vs. BPSK only.

4.2 An ML BER Union Bound for General Block Codes

If we assume an ML decoder, we can obtain some insight into the properties of general block codes—those with and without distance (i.e., $d_{min} > 1$, and $d_{min} = 1$, respectively). For an (n, κ) block code (linear or nonlinear) using an ML decoder we will show that in an AWGN channel the data bit error rate is upper bounded as

$$P_b \leq \sum_{d=d_{min}}^{n} \beta_d \cdot Q\left(\sqrt{2\frac{\kappa}{n} d \frac{E_b}{N_o}}\right) \tag{4.2}$$

where P_b is the information bit error rate and d_{min} is the minimum distance of the code. This expression is the familiar union bound for bit error rate for the encoder input bits. We are interested in the distribution of the distance between all pairs of codewords and precisely how the encoder maps these codeword error events to data block error events. This function is described by β_d. By conditioning the probability of a codeword error P_{cw} on the transmission of any one of the 2^κ codewords, we obtain

$$P_{cw} = \frac{1}{2^\kappa} \sum_{i=1}^{2^\kappa} Pr(\text{codeword error} | \boldsymbol{c}_i \text{ transmitted}). \tag{4.3}$$

Using a conditional union bound, we may write

$$\begin{aligned} Pr(\text{codeword error} | \boldsymbol{c}_i \text{ transmitted}) &\leq \sum_{\substack{j=1 \\ j \neq i}}^{2^\kappa} Pr(\text{decide } \boldsymbol{c}_j | \boldsymbol{c}_i \text{ transmitted}) \\ &= \sum_{\substack{j=1 \\ j \neq i}}^{2^\kappa} Q\left(\sqrt{2\frac{\kappa}{n} d_H(\boldsymbol{c}_i, \boldsymbol{c}_j) \frac{E_b}{N_o}}\right) \end{aligned} \tag{4.4}$$

and we have used $d_H(\boldsymbol{x}, \boldsymbol{y})$ to denote the Hamming distance between binary vectors $\boldsymbol{x}$ and $\boldsymbol{y}$. And so we obtain

$$P_{cw} \leq \frac{1}{2^{\kappa}} \sum_{i=1}^{2^{\kappa}} \sum_{\substack{j=1 \\ j \neq i}}^{2^{\kappa}} Q\left(\sqrt{2\frac{\kappa}{n} d_H(\boldsymbol{c}_i, \boldsymbol{c}_j) \frac{E_b}{N_o}}\right) \tag{4.5}$$

which represents a full union bound over the entire codebook. Since erroneously deciding $\boldsymbol{c}_j$ over $\boldsymbol{c}_i$ results in a data bit error rate of $\frac{d_H(\boldsymbol{d}_i, \boldsymbol{d}_j)}{\kappa}$, we further obtain

$$P_b \leq \frac{1}{2^{\kappa}} \sum_{i=1}^{2^{\kappa}} \sum_{\substack{j=1 \\ j \neq i}}^{2^{\kappa}} \frac{d_H(\boldsymbol{d}_i, \boldsymbol{d}_j)}{\kappa} \cdot Q\left(\sqrt{2\frac{\kappa}{n} d_H(\boldsymbol{c}_i, \boldsymbol{c}_j) \frac{E_b}{N_o}}\right) \tag{4.6}$$

Observing that we may alternatively sum the data bit errors due to a particular codeword distance error event (as opposed to the pairwise sum in (4.6)), we may equivalently write

$$\begin{aligned} P_b &\leq \frac{1}{2^{\kappa}} \sum_{d=d_{min}}^{n} \frac{2\zeta^{(d)}}{\kappa} \cdot Q\left(\sqrt{2\frac{\kappa}{n} d \frac{E_b}{N_o}}\right) \\ &= \sum_{d=d_{min}}^{n} \beta_d \cdot Q\left(\sqrt{2\frac{\kappa}{n} d \frac{E_b}{N_o}}\right) \end{aligned} \tag{4.7}$$

with

$$\beta_d = \frac{\zeta^{(d)}}{2^{k-1} \cdot \kappa}$$

and $\zeta^{(d)}$ denoting the total number of data bit errors associated with all pairs of codewords that are Hamming distance d apart, i.e.,

$$\zeta^{(d)} = \sum_{(i,j):\, d_H(\boldsymbol{c}_i, \boldsymbol{c}_j) = d} d_H(\boldsymbol{d}_i, \boldsymbol{d}_j) \tag{4.8}$$

Using the specific form (4.7) makes it easier to analyze and discuss some of the fundamental limits and properties of general nonlinear block codes by focusing on the parameter β_d. Although (4.7) is really an upper bound, it is sufficiently tight above moderate values of E_b/N_o—we later verify this with some simulation comparisons.

For codes with $d_{min} = 1$, β_1 is usually of paramount importance; it is necessary for the condition

$$\beta_1 < 1 \tag{4.9}$$

to hold in order to avoid conceding *more* than the rate loss when the code is used in the system of Figure 4.1 at moderate-to-high values of E_b/N_o. The rational behind (4.9) is that if the error rate for a BPSK-modulated (n, κ) block code is adjusted for its rate loss, the error rate expression

$$P_b = Q\left(\sqrt{2 \cdot d_{min} \frac{\kappa}{n} \frac{E_b}{N_o}}\right) \tag{4.10}$$

results. This reflects the error rate performance for a block code which concedes exactly its rate loss in the system of Figure 4.1. Observe that, for $d_{min} = 1$, the RHS of (4.10) is precisely the first term of the expression (4.7). So since there are additional terms that appear in (4.7) (compared with (4.10)) it can be seen that (4.9) is a necessary (but not sufficient) condition if the error rate in (4.7) is to be upper bounded by that of (4.10). This of course assumes equality in (4.7) which is acceptable above moderate values of E_b/N_o. But in order to increase the accuracy of a judgment on rate loss at lower values of E_b/N_o it may often be necessary to also consider the coefficient β_2 since the existence of a large number of codeword pairs at $d_H = 2$ would significantly affect the bit error rates at lower values of E_b/N_o. The rate 2/3 code used in the first motivating example has $\beta_1 = 1/2$, $\beta_2 = 1$, $\beta_3 = 1/2$. And in the second example, $\mathcal{C}_1(\boldsymbol{d})$ has $\beta_1 = 55/32$ and $\beta_2 = 86/32$ whereas $\mathcal{C}_2(\boldsymbol{d})$ has $\beta_1 = 7/32$ and $\beta_2 = 111/32$. The effect of error propagation in $\mathcal{C}_1(\boldsymbol{d})$ is obvious from its β_1 coefficient. Since these coefficients are for an ML decoder, the error rate for a MAP decoder at moderate-to-high values of E_b/N_o is expected to be roughly equivalent.

So we see that the distribution of the Hamming distance between codeword pairs is critical for codes with $d_{min} = 1$; equally critical is the how the small-distance codeword pairs are mapped to data block pairs. The ideal mapping is one for which the largest number of small-distance codeword pairs are mapped to the small-distance data block pairs, that is, the mapping producing the minimum value of β_1, β_2, and so on in that order. Within the remainder of this chapter, we refer, somewhat loosely, to this mapping as the optimum encoder.

Next we focus on a $d_{min} = 1$ rate $(n-1)/n$ code. We will show how to construct the $(0,k)$-constrained code that possesses the smallest number of $d_H = 1$ codeword pairs. We also describe the optimum block encoder for this code. For this code, we use combinatorial arguments to obtain exactly the coefficient β_1. When the resulting ML bound is compared with that for BPSK in an AWGN channel, i.e., $Q\left(\sqrt{2E_b/N_o}\right)$, it shows that, asymptotically, an ML decoder for this code is able to recover a portion of the rate loss ($10\log[n/(n-1)]$) for $n > 2$. Furthermore, for this code, plotting the ML bound also indicates that actual coding gain is possible for $n \gtrsim 8$ (we only show the plots for $n = 5$ and 17 — in Figure 4.4).

4.3 Near-optimal Rate $(n-1)/n$ $(0,k)$ Codes

The $d_{min} = 1$ codes explored in the next two sections are derived from the family of single parity check (SPC) codes of length n and so we first discuss some of the properties of these codes. Several authors in the literature ([30] and [45] for example) have used SPCs in concatenated coded systems. In particular [45] discusses a particular family of SPC codes for use as the constituent codes in a multidimensional product code; each codeword in the family having an even number of *zeros*—so the family is only $(0,k)$-constrained for odd n. In Chapter 5 we introduce

a soft-output algorithm for general block codes and show how some of these code perform in the concatenated system of Figure 3.1(a).

The construction for the $d_{min} = 2$ codes follows. We refer to the $d_{min} = 1$ code families as 'near-optimal' because we show that in channels like the AWGN channel where the performance metric is the Hamming distance, these code families perform very close to optimum—in terms of data BERs.

4.3.1 The Rate $(n-1)/n$ $(0, 2n-2)$ $d_{min} = 2$ Codes $\mathcal{C}_n$

We can partition the full set of n-tuples into two equal-size sets — $\mathcal{C}_n$ and $\mathcal{C}_n^*$. We define $\mathcal{C}_n$ to comprise of all n-tuples with odd weight and $\mathcal{C}_n^*$ all n-tuples with even weight; therefore both $\mathcal{C}_n$ and $\mathcal{C}_n^*$ are rate $(n-1)/n$ SPC codes. In addition, since the difference between any two even-weight n-tuples is also an even-weight n-tuple, $\mathcal{C}_n^*$ is linear. We also observe that, since the difference between any two n-tuples with the same parity (even or odd) is an even-weight n-tuple, both $\mathcal{C}_n$ and $\mathcal{C}_n^*$ are codes with $d_{min} = 2$. Since $\mathcal{C}_n$ contains both the weight-1 blocks $10^{(n-1)}$ and $0^{(n-1)}1$, it is a $(0, 2n-2)$ code as well. In particular, $\mathcal{C}_2$ is the well-known biphase (Manchester) code. In Facts 4.1–4.3 we give the necessary arguments that allow specification of the minimum value of the coefficient β_2. In what follows we state and prove a simple fact about $\mathcal{C}_n$.

Fact 4.1. $\mathcal{C}_n$ is the only rate $(n-1)/n$ nonlinear $(0, k)$ block code with $d_{min} = 2$; likewise, $\mathcal{C}_n^*$ is the only linear rate $(n-1)/n$ block code with $d_{min} = 2$. Furthermore, no rate $(n-1)/n$ block code can have $d_{min} > 2$.

Proof: First we observe that any rate $(n-1)/n$ nonlinear code $\overline{\mathcal{C}_n}$ such that $\overline{\mathcal{C}_n} \neq \mathcal{C}_n$ must have $d_{min} = 1$ since the exchange of one or more odd-weight codewords in $\mathcal{C}_n$ for an equal number of even-weight codewords from $\mathcal{C}_n^*$ will result in a code $\overline{\mathcal{C}_n}$ with

$d_{min} = 1$ since any codeword in $\mathcal{C}_n^*$ is $d_H = 1$ from exactly n of the codewords in $\mathcal{C}_n$. Since $\mathcal{C}_n$ and $\mathcal{C}_n^*$ divide up the space of all n-tuples into two equal-size sets, this argument is symmetric in $\mathcal{C}_n$ and $\mathcal{C}_n^*$. We can argue that the construction of *any* other rate $(n-1)/n$ block code uses the procedure just described, therefore any such code must have $d_{min} = 1$.[5] □

Fact 4.2. The number of $d_H = z_e$ (z_e even, $z_e \leq n$) codeword pairs in either $\mathcal{C}_n$ or $\mathcal{C}_n^*$ is $2^{n-2}\binom{n}{z_e}$.

Proof: In $\mathcal{C}_n$, any code word $d_H = z_e$ away from any other codeword has odd weight. Since $\mathcal{C}_n$ contains all the odd-weight n-tuples, there are $\binom{n}{z_e}$ codewords $d_H = z_e$ away from each codeword. Since there are 2^{n-1} codewords in $\mathcal{C}_n$, this results in a total of $2^{n-1}\binom{n}{z_e}$ pairings — but this product counts each pair twice. So, the number of distinct pairings is half of this, or $2^{n-2}\binom{n}{z_e}$. Since the difference between two even-weight n-tuples is also an even-weight n-tuple, and $\mathcal{C}_n^*$ contains all the even-weight n-tuples, the same result holds for $\mathcal{C}_n^*$. □

We can compactly describe the entire inter-code word distance distribution with an enumerating polynomial similar to a weight enumerating function of a linear code,

$$D(X) = \sum_{i=1}^{n} x_i X^i \tag{4.11}$$

---

[5] We also make the observation that since $\mathcal{C}_n^*$ is linear, the Singleton bound readily establishes that it must have minimum distance of 2 or less. Since $\mathcal{C}_n^*$ has minimum distance 2 it is Maximum Distance Separable. We finally point out that since the codes $\mathcal{C}_n$ and $\mathcal{C}_n^*$ have identical distance properties ($\mathcal{C}_n$ is *the* single coset code of $\mathcal{C}_n^*$), we may also infer that $\mathcal{C}_n$ also has maximal minimum distance.

where X is indeterminate and x_i is the number of code word pairs that are $d_H = i$ apart. Using this notation, we have, for $\mathcal{C}_n$,

$$D_{\mathcal{C}_n}(X) = 2^{n-2}\left[\binom{n}{2}X^2 + \binom{n}{4}X^4 + \cdots + \binom{n}{\lfloor n \rfloor_e}X^{\lfloor n \rfloor_e}\right] \tag{4.12}$$

with $\lfloor x \rfloor_e$ denoting the largest even integer in x. $\mathcal{C}_n^*$ obviously has exactly the same enumerating polynomial. Note also that, since $\mathcal{C}_n^*$ is MDS, its complete weight distribution is known [74].

Next, a consideration of the manner in which the encoder maps data blocks to codewords requires that we be able to specify the number of $d_H = j$ pairs in a full set of $(n-1)$-tuples. The following fact provides the required result.

Fact 4.3. The number of weight-j pairs in a full set of n-tuples is $2^{n-1}\binom{n}{j}$.

Proof: There are $\binom{n}{j}$ n-tuples $d_H = j$ away from each n-tuple. Since there are 2^n n-tuples, an identical argument to that used in the proof of Fact 4.1 provides the required result. □

The code $\mathcal{C}_n$ has $d_{min} = 2$ so $\beta_1 = 0$. From Fact 4.1 there are $2^{n-2}\binom{n}{2}$ $d_H = 2$ codeword pairs in $\mathcal{C}_n$; from Fact 4.3, there are only $2^{n-2}(n-1)$ $d_H = 1$ data block pairs. Therefore we will never be able to assign all the $d_H = 2$ codeword pairs to $d_H = 1$ data block pairs. By introducing another viewpoint to the structure of these simple codes, we can easily obtain the minimum value of the coefficient β_2; in addition we will also show that the systematic encoder achieves this minimum value of β_2. So the systematic encoder is optimum.

We first note that the entire collection of n-tuples (taken as a $2^n \times n$) matrix can be assembled recursively from the full set of $(n-1)$-tuples by a simple augment-

and-stack operation. Each $(n-1)$-tuple (e.g. abc) contributes the two n-tuples $abc0$ and $abc1$ in this fashion, for $n = 4$ for example;

$$\left.\begin{array}{ccc} a & b & c \\ d & e & f \\ & \vdots & \end{array}\right\} \Rightarrow \left\{\begin{array}{cccc} a & b & c & 0 \\ a & b & c & 1 \\ d & e & f & 0 \\ d & e & f & 1 \\ \vdots & \vdots & \vdots & \vdots \end{array}\right. \tag{4.13}$$

This observation, together with the fact that $\mathcal{C}_n$ consists of the odd-parity half of the set of n-tuples immediately suggests the following systematic encoder; simply append either a *zero* or a *one* to the data block to obtain odd parity.

We assert that this simple mapping is also optimum in that it assigns as many $d_H = 2$ codeword pairs to $d_H = 1$ data block pairs as is possible, and assigns the rest of the $d_H = 2$ codeword pairs to $d_H = 2$ data block pairs. In other words, the mapping results in the minimum possible sum of data bit errors resulting from $d_H = 2$ codeword error events (the coefficient $\zeta^{(2)}$). We formally make the following assertion;

Assertion 4.1. The mapping $\chi : \boldsymbol{d} = [d_0, d_1, \cdots, d_{n-2}] \mapsto \boldsymbol{c} = [\boldsymbol{d}|x]$; $x = \overline{\sum_{i=0}^{n-2} \oplus\, d_i}$ (with the operator $\sum\!\!\!\!\oplus$ implementing the *modulo* 2 sum) results in the minimum value of $\zeta^{(2)}$.[6]

We argue two points to support this assertion. First, χ ensures that any $d_H = 1$ data block pair $(\boldsymbol{d}_i, \boldsymbol{d}_j)$ corresponds to a $d_H = 2$ codeword pair $(\boldsymbol{c}_i, \boldsymbol{c}_j)$. To see this, let

$$\begin{array}{l} \boldsymbol{d}_i = a, b, c, \cdots, y \\ \boldsymbol{d}_j = a, b, c, \cdots, \overline{y} \end{array} \tag{4.14}$$

be a $d_H = 1$ data block pair. Without loss of generality, let $\boldsymbol{d}_i$ have even parity and therefore $\boldsymbol{d}_j$ odd parity; according to χ,

---

[6] $\boldsymbol{a}|\boldsymbol{b}$ is used to refer to the concatenation of $\boldsymbol{a}$ and $\boldsymbol{b}$.

$$\begin{aligned} \boldsymbol{c}_i &= a, b, c, \cdots, y, 1 \\ \boldsymbol{c}_j &= a, b, c, \cdots, \overline{y}, 0 \end{aligned} \tag{4.15}$$

and clearly $d_H(\boldsymbol{c}_i, \boldsymbol{c}_j) = 2$. So every $d_H = 1$ data block pair is used by a $d_H = 2$ codeword pair.

Second, if $d_H(\boldsymbol{c}_i, \boldsymbol{c}_j) = 2$, then $d_H(\boldsymbol{d}_i, \boldsymbol{d}_j) \in \{1, 2\}$. To see this, let us define, consistent with χ,

$$\left.\begin{aligned} \boldsymbol{c}_i &= \boldsymbol{d}_i | x \\ \boldsymbol{c}_j &= \boldsymbol{d}_j | y \end{aligned}\right\} \text{with } d_H(\boldsymbol{c}_i, \boldsymbol{c}_j) = 2. \tag{4.16}$$

There are two cases involved in specifying $d_H(\boldsymbol{d}_i, \boldsymbol{d}_j)$.

(i) $x = y, \Rightarrow d_H(\boldsymbol{d}_i, \boldsymbol{d}_j) = 2.$

(ii) $x = \overline{y}, \Rightarrow d_H(\boldsymbol{d}_i, \boldsymbol{d}_j) = 1.$

So every $d_H = 2$ codeword pair is mapped to either a $d_H = 1$ or a $d_H = 2$ data block pair. χ is therefore optimum. We also point out that the foregoing arguments make it clear that the actual location of the parity bit within the n-bit codeword does not affect any of the results. The runlength k parameter is also not affected by the location of the parity bit, for the operation of moving the parity bit to another location within the codeword is equivalent to adding a fixed n-tuple of weight two to each codeword. This does not change the code since the code has a single nontrivial coset whose representative is any of the codewords (and no codeword can be weight-two).

So, from Fact 4.3, there are $2^{n-2}(n-1)$ $d_H = 2$ codeword pairs mapped to $d_H = 1$ data block pairs. The remaining $2^{n-2}\binom{n}{2} - [2^{n-2}(n-1)]$ $d_H = 2$ codeword pairs are mapped to $d_H = 2$ data block pairs. Therefore,

$$\begin{aligned} \zeta_{min}^{(2)} &= [2^{n-2}(n-1)](1) + \left\{2^{n-2}\binom{n}{2} - [2^{n-2}(n-1)]\right\}(2) \\ &= 2^{n-2}(n-1)^2 \end{aligned} \tag{4.17}$$

and so $\beta_2^{min} = n - 1$. This indicates that the 3 dB distance gain provided by this code will always be offset by an encoder, resulting in a net coding gain of less than 3 dB. For $n = 2$ we see that an ML decoder for the biphase code would break even in AWGN; plotting the bound with n as parameter (not shown) also shows that, asymptotically, net coding gain would be realized for $n > 2$. Figure 4.4 shows the two-term ML approximation for $n = 5$ and 17; note that asymptotically they differ by only about 0.5 dB.

4.3.2 The Rate $(n-1)/n$, $(0, 2n-3)$, $d_{min} = 1$ Codes C'_n

Here we show that the $d_{min} = 1$ rate $(n-1)/n$ $(0, k)$ block code C'_n possessing the smallest number of $d_H = 1$ codeword pairs has $k = 2n - 3$. We continue using the definitions of C_n and C_n^* from the previous section.

Since C_n^* contains all the even-weight n-tuples, replacing any single codeword $\boldsymbol{c}_r$ in C_n with one from C_n^* will produce a new rate $(n-1)/n$ code C'_n with $d_{min} = 1$. C'_n thus has $2^{n-1} - 1$ odd-weight codewords and a single even-weight codeword $\boldsymbol{c}'_e$. It is the rate $(n-1)/n$, $d_{min} = 1$ code containing the fewest number of $d_H = 1$ codeword pairs. We need to know how many $d_H = 1$ codeword pairs there are in C'_n.

Fact 4.4. The minimum number of $d_H = 1$ codeword pairs in C'_n is $n - 1$.

Proof: We first observe that every codeword $d_H = 1$ away from an even-weight codeword has odd weight. Therefore, at most n of the codewords in C'_n are $d_H = 1$ away from $\boldsymbol{c}'_e$. But if $\boldsymbol{c}_r$ is one of these codewords, then there are only $n - 1$ $d_H = 1$ pairs in C'_n. Therefore $n - 1$ is the minimum number of $d_H = 1$ pairs in C'_n. □

If c_r is one of the weight-1 codewords—specifically if

$$c_r = 0^{(n-1)}1 \tag{4.18}$$

then we have reduced the runlength k parameter by 1 so $\mathcal{C}'_n$ is a $(0, 2n-3)$ code. We may also need to know the number of $d_H = 2$ pairs in $\mathcal{C}'_n$.

Fact 4.5. The number of $d_H = 2$ pairs in $\mathcal{C}'_n$ is $\binom{n}{2}[2^{n-2} - 1]$.

Proof: From Fact 4.1 there are $\binom{n}{2}2^{n-2}$ $d_H = 2$ pairs in $\mathcal{C}_n$. Since $\boldsymbol{c}_r$ was $d_H = 2$ away from exactly $\binom{n}{2}$ codewords in $\mathcal{C}_n$ (and $\boldsymbol{c}'_e$ cannot contribute to any $d_H = 2$ pairs in $\mathcal{C}_n$ because it has even weight), we are left with $\binom{n}{2}$ less $d_H = 2$ pairs. The result follows. □

We can now easily compute $\zeta^{(1)}$. For the code $\mathcal{C}'_n$ possessing the least number of $d_H = 1$ codeword pairs, we can certainly design the encoder such that each $d_H = 1$ codeword pair corresponds to a pair of data words that are also $d_H = 1$ apart — so they can contribute only a single data bit error. To show that this is true consider the following. In $\mathcal{C}'_n$, each of the $n-1$ $d_H = 1$ codeword pairs is formed between $\boldsymbol{c}'_e$ and one of the other odd-parity codewords. So without loss of generality, the complete set of $d_H = 1$ codeword pairs has the form

$$\{(\boldsymbol{c}'_e, \boldsymbol{c}'_i)\},\ i = 1, 2, \cdots, n-1 \tag{4.19}$$

where i is simply being used to arbitrarily index the $n-1$ codewords. In the full set of $(n-1)$-tuples, it will always be possible to assemble a set of $n-1$ data words with exactly the same form as (19) since there are exactly $n-1$ $(n-1)$-tuples $d_H = 1$ away from any given data word. So to recapitulate, the $n-1$ $d_H = 1$ codeword pairs in $\mathcal{C}'_n$ need not result in more than $n-1$ data bit errors, i.e., $\zeta^{(1)}_{min} = n-1$, and therefore from (7),

$$\begin{aligned} \beta_1^{min} &= \frac{2(n-1)}{(n-1)\cdot 2^{n-1}} \\ &= \frac{1}{2^{n-2}}. \end{aligned} \tag{4.20}$$

It is well known that for moderate-to-high values of E_b/N_o the first term in the ML union bound is usually the only significant one. However, due to the large number of $d_H = 2$ codeword pairs in $\mathcal{C}'_n$ we have plotted the first two terms of the ML bound in Figure 4.4 for $n = 5$ and 17. We have assumed an average value for the coefficient β_2 since it depends on the *full* encoder[7]. Using Fact 4.3 we can evaluate the average Hamming distance between data block pairs for a randomly chosen encoder. The expected value of the Hamming distance between a pair of n-tuples chosen at random from the full set of all n-tuples is easily shown to be

$$\mathrm{E}[d_H(\boldsymbol{d}_{n_i}, \boldsymbol{d}_{n_j})] = \frac{1}{2^n - 1} \sum_{m=1}^{n} m \binom{n}{m}. \tag{4.21}$$

So, for a given (n, κ) code with known number γ_i of $d_H = i$ codeword pairs, we can easily estimate the expected value of the coefficient $\zeta^{(i)}$ for an average encoder by taking the product $\gamma_i \mathrm{E}[d_H(\boldsymbol{d}_{\kappa_i}, \boldsymbol{d}_{\kappa_j})]$. We have used this average value for $\zeta^{(2)}$ for $\mathcal{C}'_5$ and $\mathcal{C}'_{17}$ in Figure 4.4. The curves indicate that partial rate loss recovery is possible for $n = 5$ (the corresponding curve lies less than $10\log(5/4) = 1$ dB to the right of the BPSK curve); and for $n = 17$, a coding gain (approaching 2 dB) is actually possible. Although not shown, the lowest n for which net coding gain is realized is 8. The rate 2/3 code used to illustrate the first motivating example is actually $\mathcal{C}'_3$.

The actual encoder function for this code is not unique. For example, $\boldsymbol{c}'_e$could initially be assigned to any of the 2^{n-1} data blocks; this then fixes many of the other mappings.

---

[7] In the foregoing analysis we have only specified the portion of the encoder which focusses on mapping $d_H = 1$ codeword pairs to data block pairs—there are, in fact, $2^\kappa - n$ unspecified codewords/data blocks which results in a total of $(2^\kappa - n)!$ codeword-to-data block mappings.

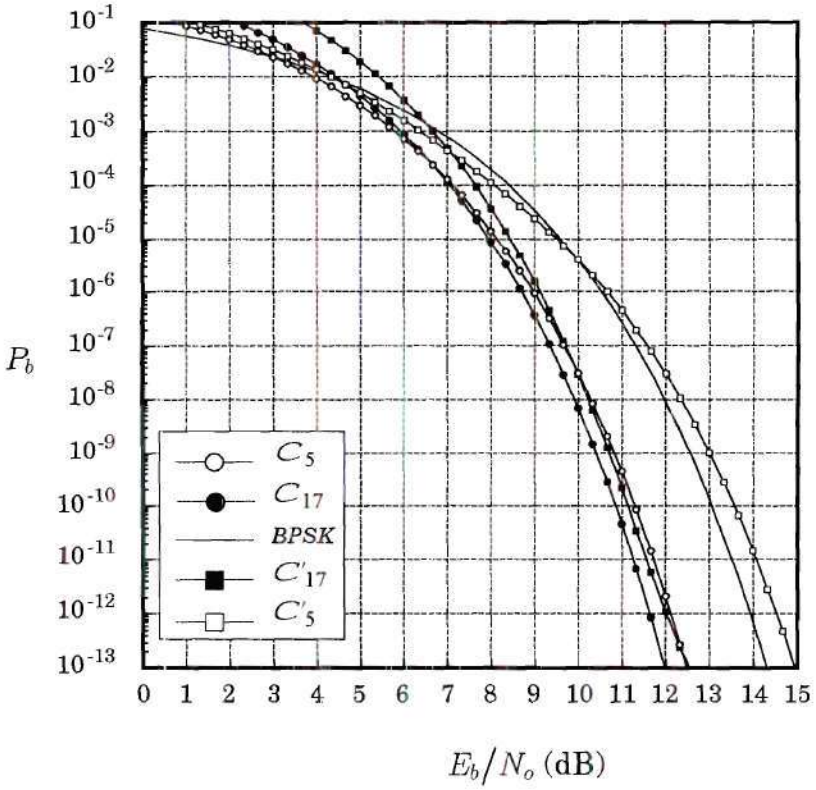


Figure 4.4. Two-term ML bounds for $\mathcal{C}_n$ and $\mathcal{C}'_n$ for $n = 5$ and 17. Also shown is the error rate for BPSK.

4.3.3 Two Slightly Tighter Rate $(n-1)/n$, $(0,k)$, $d_{min}=1$ Codes $\mathcal{C}_n''$ and $\mathcal{C}_n^{4\prime}$

We can extend the tacit construction of the previous section to obtain slightly tighter $(0,k)$ codes with the least possible number of $d_H = 1$ codeword pairs. We observe a tradeoff between the runlength k parameter and the coefficient β_1— the tighter the $(0,k)$ constraint (the smaller k is), the larger β_1 is by way of an increased number of $d_H = 1$ codeword pairs. So as we progressively 'dilute' $\mathcal{C}_n$ with even-parity codewords in order to reduce k, the number of $d_H = 1$ codeword pairs grows. It thus becomes clear that the penalty for a smaller k is the requirement of a larger n in order to achieve net coding gain—or equivalently, the same n provides a smaller asymptotic rate loss recovery.

Omitting most of the details (refer to appendix A.1 for the details), the two codewords

$$\begin{aligned} \boldsymbol{c}_{r1} &= 0^{(n-1)}1 \\ \boldsymbol{c}_{r2} &= 10^{(n-1)} \end{aligned} \tag{4.22}$$

in $\mathcal{C}_n$ are replaced with

$$\begin{aligned} \boldsymbol{c}''_{e1} &= 0^{(n-2)}11 \\ \boldsymbol{c}''_{e2} &= 10^{(n-2)}1 \end{aligned} \tag{4.23}$$

respectively, to obtain $\mathcal{C}_n''$ which is the $(0, 2n-4)$ code with the least number of $d_H = 1$ codeword pairs which is equal to $2n-3$. In Appendix A.1 it is shown that all the $d_H = 1$ codeword pairs can be assigned to $d_H = 1$ data block pairs. Since the essence of the optimum encoder is in the *patterns* between data blocks that correspond to different codewords, it can easily be shown that a total of $\binom{n-1}{2}2^{n-1}$ equivalent mappings exist for a given code of length n — a rather sizable number. This large number allows quite a bit of design freedom in choosing a simpler encoder (perhaps by trial and error). So we can conclude that, for $\mathcal{C}_n''$, the complete set of the

$d_H = 1$ codeword pairs need not cause more than a total of $2n - 3$ data bit errors. Therefore, for this family of codes $\zeta_{min}^{(1)} = 2n - 3$, and so,

$$\beta_1^{min} = \frac{2n-3}{(n-1)2^{n-2}},\ n \geq 3, \text{for } \mathcal{C}_n'' \tag{4.24}$$

and for $n \geq 3$ at least partial rate loss recovery is available; coding gain, though, is unavailable until $n \geq 9$. The code $\mathcal{C}_2(\boldsymbol{d})$ in the second motivating example is actually $\mathcal{C}_5''$. Since the encoder is partially systematic (the first three coordinates of $\boldsymbol{d}$ are also the first three coordinates of $\mathcal{C}_5''(\boldsymbol{d})$) we need store only the last two bits of each codeword.[8]

This construction may be extended a step further to yield the family of $(0, 2n-6)$ codes $\mathcal{C}_n^{4\prime}$ with the least number of $d_H = 1$ codeword pairs. See Appendix A.2 for details. The arguments are similar to those used previously. For this case, the four codewords

$$\begin{aligned} \boldsymbol{c}_{r1} &= 0^{(n-1)}1 \\ \boldsymbol{c}_{r2} &= 10^{(n-1)} \\ \boldsymbol{c}_{r3} &= 0^{(n-2)}10 \\ \boldsymbol{c}_{r4} &= 010^{(n-2)} \end{aligned} \tag{4.25}$$

in $\mathcal{C}_n$ are replaced with

$$\begin{aligned} \boldsymbol{c}_{e1}^{4\prime} &= 0^{(n-3)}101 \\ \boldsymbol{c}_{e2}^{4\prime} &= 10^{(n-2)}1 \\ \boldsymbol{c}_{e3}^{4\prime} &= 10^{(n-3)}10 \\ \boldsymbol{c}_{e4}^{4\prime} &= 010^{(n-4)}10 \end{aligned} \tag{4.26}$$

respectively. This set of $\{\boldsymbol{c}_{ei}^{4\prime}\}$ is not unique but it certainly produces a code $\mathcal{C}_n^{4\prime}$ with the least number of $d_H = 1$ codeword pairs. The rule is to choose $\boldsymbol{c}_{ei}^{4\prime}$ so that two of its $d_H = 1$ neighbors are $\boldsymbol{c}_{ri}$ and $\boldsymbol{c}_{r(i-1)}$ while simultaneously satisfying the target

---

[8] No such simplification is possible with the previous example involving $\mathcal{C}_4''$.

$(0, 2n-6)$ constraint. We would like the $\{\boldsymbol{c}_{ei}^{4\prime}\}$ to have as many $d_H = 1$ neighbors in common (all of these neighbors are odd-parity n-tuples in $\mathcal{C}_n$, some of which are removed as the $\{\boldsymbol{c}_{ri}\}$) and it can be demonstrated that selecting them all to be weight-2 achieves this. The resulting code $\mathcal{C}_n^{4\prime}$ possesses a total of $4n-7$ $d_H = 1$ codeword pairs which is the minimum possible number. It can also be demonstrated that, for $n \geq 6$, each of these $4n-7$ codeword pairs can be assigned by an encoder to a $d_H = 1$ data block pair. For $n \in \{4, 5\}$ there are not enough $d_H = 1$ data block pairs to support all of these codeword pairs, so we have,

$$\left.\begin{aligned} \beta_1^{min} &= \frac{4n-7}{(n-1)2^{n-2}}, && n \geq 6 \\ &= \frac{15}{4 \cdot 2^3}, && n = 5 \\ &= \frac{8}{3 \cdot 2^2}, && n = 4 \end{aligned}\right\} \text{ for } \mathcal{C}_n^{4\prime}. \tag{4.27}$$

Here partial rate loss recovery is possible for $n \geq 4$ and coding gain for $n \geq 10$.

It is undoubtedly possible to continue this 'successive dilution' of $\mathcal{C}_n$ to yield tighter $(0, k)$ codes with more $d_H = 1$ codeword pairs, but it becomes increasingly difficult and tedious to show what the construction looks like and to determine the optimal codeword-to-data block mappings. We therefore conclude this section by tabulating some of the properties of the code families from this and previous subsections in Table 4.3. In Figure 4.5 we show a comparison between a simulation of the system in Figure 4.1 and the ML approximations derived in this and the previous section for the code families; this is shown for $n = 5$. The simulations show very good agreement with the 2-term ML bound; the tradeoff between k and error rate is evident. We reiterate that we have used an average value for the second term in the ML bound.

Since all of these block $(0, k)$ codes ($\mathcal{C}_n$, $\mathcal{C}'_n$, $\mathcal{C}''_n$, and $\mathcal{C}^{4\prime}_n$) have rather large values of k, they are unlikely to be used for large values of n, i.e., $n > 9$. For such relatively small codebook cardinality a true ML or MAP decoder may be practical. The ML decoder performance we have discussed so far highlights the fundamental tradeoff between the k parameter for block $(0, k)$ codes and their distance properties. Even though these codes do not really possess any error-correction ability (due to the lack of a minimum distance of at least three), the ML performance curves show that it is possible for SISO decoders for these codes to provide a modest net coding gain compared to an uncoded system. In the next chapter we show how some of these simple codes perform when used together with turbo codes in an AWGN channel.

Table 4.3. Some properties of some rate $(n-1)/n$, $(0, k)$ codes.

Code	k	# $d_H = 1$ *pairs*	β_1^{min}	*n for coding gain*	*n to recover rate loss*	*comments*
$\mathcal{C}_n$	$2n-2$	0	0	3	3	The only possible $(0, k)$ $d_H = 2$ codes. Systematic encoder is optimal.
$\mathcal{C}'_n$	$2n-3$	$n-1$	$\frac{1}{2^{n-2}}, n \geq 2$	8	3	code with fewest possible number of $d_H = 1$ codeword pairs.
$\mathcal{C}''_n$	$2n-4$	$2n-3$	$\frac{2n-3}{(n-1)2^{n-2}}, n \geq 3$	9	3	Contains only 2 even-weight codewords
$\mathcal{C}^{4\prime}_n$	$2n-6$	$4n-7$	$\frac{4n-7}{(n-1)2^{n-2}}$, $n \geq 6$ $\frac{15}{4 \cdot 2^3}$, $n = 5$ $\frac{8}{3 \cdot 2^2}$, $n = 4$	10	4	Contains only 4 even-weight codewords

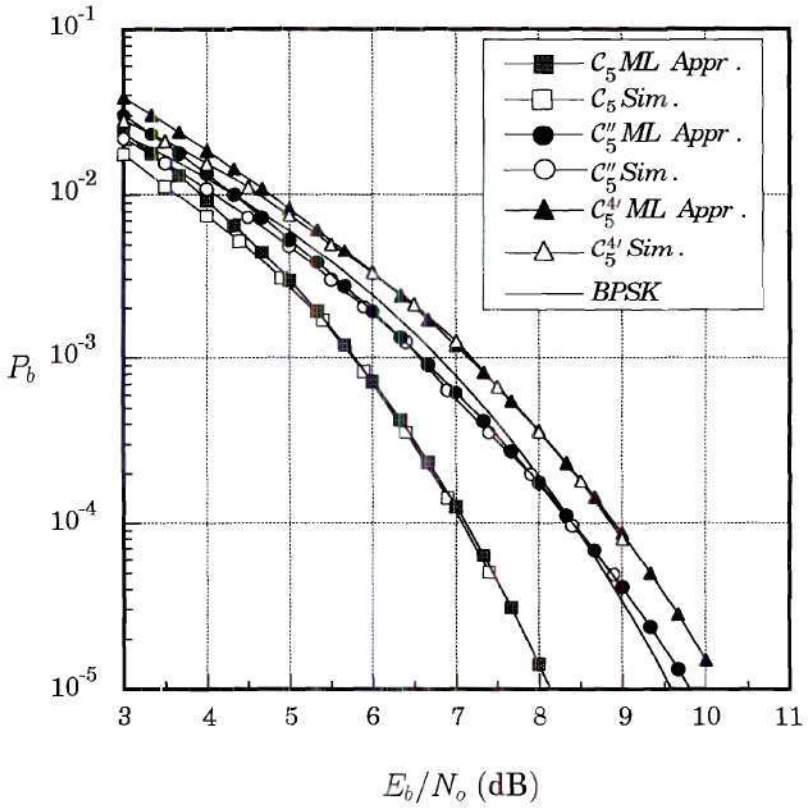


Figure 4.5. 2-term ML bound and simulated performance of an ML decoder in AWGN for $\mathcal{C}_5$, $\mathcal{C}_5''$, and $\mathcal{C}_5^{4\prime}$.

CHAPTER V

TURBO CODES CASCADED WITH RATE $(n-1)/n$ BLOCK CODES ON $(0,k)$-CONSTRAINED CHANNELS

In this chapter, we look at the performance of the standard concatenation of Figure 3.1(a) on both the AWGN and PR channels. We limit the treatment of the constrained codes to the $(0,k)$ constraint; in addition, we also only consider such constrained codes with rate $(n-1)/n$. Specifically, we show how the two different kinds of channels favor different properties of rate $(n-1)/n$, $(0,k)$-constrained codes. It is well known that the Hamming distance of block (constrained) codes is the important metric in AWGN—we showed how to construct several good block $(0,k)$-constrained codes for this metric in the preceding chapter. We should expect that, in the concatenated system of 3.1(a), these codes will do well in AWGN channels. We show that the ML performance of these codes translates predictably into rate loss recovery when they are used in the standard concatenated system of Figure 3.1(a). But we demonstrate that, when the channel is a precoded PR target, there are better-performing (in terms of BER) rate $(n-1)/n$, $(0,k)$-constrained codes than these. If we are to consider the configuration in Figure 3.1(a) we will need a SISO decoder for the constrained code (a 'soft demodulator'); we therefore give a method for the computation of the log APP (LAPP) ratio for the input bits of a general linear or nonlinear block code (e.g. a $(0,k)$-constrained block code).

5.1 LAPP Computation for General Block Codes

This derivation and discussion apply, generally, to any linear or nonlinear block code. In order to be more precise we will use the expression 'soft information' when referring to AWGN-corrupted BPSK symbols (or an estimate of such symbols) as would be obtained from a sampled matched filter—and 'LAPP ratio' according to the definition (5.1). We assume a general (n, κ) block code C for which there is a one-to-one mapping from each distinct κ-tuple over GF(2) ("data bits" or "data block") to one of the 2^κ codewords of length n (also over GF(2)). First we consider a general block code.

5.1.1 General Form of LAPP Ratio

Let one of the 2^κ input blocks be represented by $\boldsymbol{d}^{(i)} = (d_0^{(i)}, d_1^{(i)}, \cdots, d_{\kappa-1}^{(i)})$, $i \in \{0, 1, \cdots, 2^\kappa - 1\}$ and the corresponding codeword by $\boldsymbol{c}^{(i)} = (c_0^{(i)}, c_1^{(i)}, \cdots, c_{n-1}^{(i)})$. Assume $c^{(i)}$ is BPSK-modulated and transmitted over a noisy channel resulting in the noisy received vector $\boldsymbol{r} = (r_0, r_1, \cdots, r_{n-1})$ from which we would like to compute the LAPP ratio for an input bit d_j, $j \in \{0, 1, \cdots, \kappa - 1\}$ defined by

$$L(d_j) = \log \frac{Pr(d_j = 1|\boldsymbol{r})}{Pr(d_j = 0|\boldsymbol{r})} \tag{5.1}$$

Invoking Bayes' theorem, we obtain

$$\begin{aligned} L(d_j) &= \log \frac{p(r|d_j = 1)Pr(d_j = 1)}{p(r|d_j = 0)Pr(d_j = 0)} + \log \frac{Pr(d_j = 1)}{Pr(d_j = 0)} \\ &= \log \frac{p(r|d_j = 1)}{p(r|d_j = 0)} + L^{apr}(d_j) \\ &= \log \frac{\sum_{i \in \mathcal{S}_1(j)} p(r|c^{(i)} \text{ transmitted})}{\sum_{i \in \mathcal{S}_0(j)} p(r|c^{(i)} \text{ transmitted})} + L^{apr}(d_j). \end{aligned} \tag{5.2}$$

$L^{apr}(d_j)$ is the usual *a priori* term; it is set equal to zero if the encoder input bits are equiprobable. In the ensuing discussion we ignore it, noting that it can be added where necessary. The disjoint sets $\mathcal{S}_1(j)$and $\mathcal{S}_0(j)$ (for given j) contain the indices i for which $c^{(i)}$ corresponds to a data block $d^{(i)}$ for which $d_j^{(i)} = 1$, and $d_j^{(i)} = 0$, respectively; clearly $\mathcal{S}_0(j) \cup \mathcal{S}_1(j) = \{0, 1, \cdots, 2^\kappa - 1\}$. The last line of (5.2) encapsulates both the encoder mapping of information bits to coded bits and the distance properties of the block code. In the most general sense, the encoder function can be expressed as a set of n Boolean functions (one for each codeword bit) each of κ variables (the data bits). The expression (5.2) is exact although not practical for code books with large cardinality, since the complexity of the computation escalates exponentially with κ (exactly as in the case for an ML decoder). This requirement for practical feasibility coincides with that for use of the code families discussed in the previous chapter except that the limits for those codes are set by the length of the runlength k parameter — in both cases values of n much larger than 9 or so become impractical. It is possible to obtain a suboptimal estimate of the soft $L(d_j)$ by using some method to significantly reduce the size of the set of codewords over which Euclidean distances are computed. Chase [16] demonstrated that there are several algorithms for selecting a good subset of the code book which drastically reduces the required computational burden in exchange for a moderate performance penalty. However, we have observed that, while the Chase algorithms produce reliability values with a very reliable sign, the actual reliability magnitudes are exaggerated, and when passed to a turbo decoder lead to poor overall performance (significant additional coding rate loss).[9] Also responsible for this poor performance, no doubt, is the fact

[9] For example, the variant of the Chase algorithm for which (5.2) is computed only over codewords within $d_H = 1$ of the hard estimate of $\boldsymbol{r}$ does not discernibly sacrifice any error rate

that with these suboptimal solutions the resulting distribution of $L(d_j)$ (conditioned on knowledge of d_j) is no longer Gaussian as it is for the optimal solution (5.2).

In Figure 3.1(a) the RLL decoder needs to pass soft information to a soft-input ECC; when the channel noise is Gaussian and white, (5.2) takes the form

$$L(d_j) = \log \frac{\sum_{i \in \mathcal{S}_1(j)} \exp\left\{ -\frac{1}{2\sigma^2} \sum_{m=0}^{n-1} \left[r_m - M\left(c_m^{(i)}\right) \right]^2 \right\}}{\sum_{i \in \mathcal{S}_0(j)} \exp\left\{ -\frac{1}{2\sigma^2} \sum_{m=0}^{n-1} \left[r_m - M\left(c_m^{(i)}\right) \right]^2 \right\}}, \tag{5.3}$$

with $M\left(c_m^{(i)}\right)$ representing the modulation function. For example, with normalized BPSK modulation, we would write

$$M\left(c_m^{(i)}\right) = 2c_m^{(i)} - 1. \tag{5.4}$$

Although apparently difficult to show theoretically from (5.3), we have observed from repeated simulation of this expression on a large number of block codes that $L(d_j)$ is also conditionally Gaussian given d_j. So as an approximation, we could view $L(d_j)$ as the log likelihood ratio (LLR) for the bit d_j. This bit is assumed to have been BPSK-modulated and sent over a fictitious AWGN channel resulting in $\widehat{r}_j$ so that we could write,

$$L(d_j) \approx \frac{2}{\sigma^2} \widehat{r}_j. \tag{5.5}$$

This permits a simple conversion between the LAPP ratio $L(d_j)$ computed from (5.2) above, and $\widehat{r}_j$ which is the equivalent soft information actually passed to the turbo decoder [64], [4].

---

performance compared to an ML decoder in the system of Fig. 4.1. But use of this algorithm for LAPP computation results in an additional coding loss of 0.5 dB compared to the optimal (5.2) for the full system of Fig. 5.1(a) with $\mathcal{C}_5''$.

5.1.2 Separation into Extrinsic and Systematic Parts

It is also possible to express $L(d_j)$ explicitly as a function of the LLRs of the code bits $\{c_m\}$. This casts $L(d_j)$ in a form that is very similar to that often seen for the case when the BCJR algorithm is operated over a trellis (a quotient of sums of exponentials). As we show in the next section, this alternative form also facilitates an evaluation of the complexity of the computation. If r_m is an AWGN-corrupted version of a BPSK-modulated generic code bit c_m then the LLR of c_m, $\mathcal{L}(c_m)$, is just

$$\mathcal{L}(c_m) = \frac{2}{\sigma^2} r_m. \tag{5.6}$$

We may then, equivalently write (5.3) as

$$L(d_j) = \log \frac{\sum_{i \in \mathcal{S}_1(j)} \exp\left[\sum_{m:c_m^{(i)}=1} \mathcal{L}(c_m)\right]}{\sum_{i \in \mathcal{S}_0(j)} \exp\left[\sum_{m:c_m^{(i)}=1} \mathcal{L}(c_m)\right]}. \tag{5.7}$$

This is obtained by observing that, using (5.4), we may write (5.3) as

$$\begin{aligned} L(d_j) &= \log \frac{\sum_{i \in \mathcal{S}_1(j)} \exp\left[\frac{1}{\sigma^2} \sum_{m=0}^{n-1} r_m \left(2c_m^{(i)} - 1\right)\right]}{\sum_{i \in \mathcal{S}_0(j)} \exp\left[\frac{1}{\sigma^2} \sum_{m=0}^{n-1} r_m \left(2c_m^{(i)} - 1\right)\right]} \\ &= \log \frac{\sum_{i \in \mathcal{S}_1(j)} \exp\left[\frac{1}{\sigma^2} \boldsymbol{r} \cdot \left(2\boldsymbol{c}^{(i)} - \mathbf{1}\right)\right]}{\sum_{i \in \mathcal{S}_0(j)} \exp\left[\frac{1}{\sigma^2} \boldsymbol{r} \cdot \left(2\boldsymbol{c}^{(i)} - \mathbf{1}\right)\right]} \end{aligned} \tag{5.8}$$

and continuing,

$$L(d_j) = \log \frac{\sum_{i \in \mathcal{S}_1(j)} \exp\left(\frac{2}{\sigma^2} \boldsymbol{r} \cdot \boldsymbol{c}^{(i)}\right) \cdot \exp\left(-\frac{1}{\sigma^2} \sum_m r_m\right)}{\sum_{i \in \mathcal{S}_0(j)} \exp\left(\frac{2}{\sigma^2} \boldsymbol{r} \cdot \boldsymbol{c}^{(i)}\right) \cdot \exp\left(-\frac{1}{\sigma^2} \sum_m r_m\right)}, \tag{5.8}$$

and we have used 1 to denote a row vector of '1's of the appropriate size, and $\boldsymbol{a} \cdot \boldsymbol{b}$ is the inner product of $\boldsymbol{a}$ and $\boldsymbol{b}$. Since $\exp\left(-\frac{1}{\sigma^2}\sum_m r_m\right)$ is a constant, we obtain

$$L(d_j) = \log \frac{\sum_{i \in \mathcal{S}_1(j)} \exp\left(\frac{2}{\sigma^2}\boldsymbol{r} \cdot \boldsymbol{c}^{(i)}\right)}{\sum_{i \in \mathcal{S}_0(j)} \exp\left(\frac{2}{\sigma^2}\boldsymbol{r} \cdot \boldsymbol{c}^{(i)}\right)}, \tag{5.9}$$

which is identical to (5.7).

For a nonsystematic block code, the RHS of (5.7) or (5.9) yields the same result as the RHS of (5.3), and since the code is nonsystematic $L(d_j)$ can also be viewed as the extrinsic information for d_j (since there is no 'channel value' to subtract out).

For a systematic block code, the RHS of (5.9) separates into a systematic part (channel value) and an extrinsic part. To see this, let us assume, without loss of generality, that the systematic coordinates occupy the first κ positions in the code. Also, let $\boldsymbol{a}\backslash a_i$ denote the vector $\boldsymbol{a}$ with element a_i deleted. Then

$$\boldsymbol{r} \cdot \boldsymbol{c}^{(i)} = \begin{cases} r_j + (\boldsymbol{r}\backslash r_j) \cdot \left(\boldsymbol{c}^{(i)}\backslash c_j^{(i)}\right), & \forall i \in \mathcal{S}_1(j) \\ (\boldsymbol{r}\backslash r_j) \cdot \left(\boldsymbol{c}^{(i)}\backslash c_j^{(i)}\right), & \forall i \in \mathcal{S}_0(j) \end{cases} \tag{5.10}$$

so that

$$L(d_j) = \log \frac{\exp\left(\frac{2}{\sigma^2}r_j\right) \cdot \sum_{i \in \mathcal{S}_1(j)} \exp\left[\frac{2}{\sigma^2}(\boldsymbol{r}\backslash r_j) \cdot \left(\boldsymbol{c}^{(i)}\backslash c_j^{(i)}\right)\right]}{\sum_{i \in \mathcal{S}_0(j)} \exp\left[\frac{2}{\sigma^2}(\boldsymbol{r}\backslash r_j) \cdot \left(\boldsymbol{c}^{(i)}\backslash c_j^{(i)}\right)\right]} \tag{5.11}$$

and continuing,

$$\begin{aligned} L(d_j) &= \frac{2}{\sigma^2}r_j + \log \frac{\sum_{i \in \mathcal{S}_1(j)} \exp\left[\frac{2}{\sigma^2}(\boldsymbol{r}\backslash r_j) \cdot \left(\boldsymbol{c}^{(i)}\backslash c_j^{(i)}\right)\right]}{\sum_{i \in \mathcal{S}_0(j)} \exp\left[\frac{2}{\sigma^2}(\boldsymbol{r}\backslash r_j) \cdot \left(\boldsymbol{c}^{(i)}\backslash c_j^{(i)}\right)\right]} \\ &= L^{sys}(d_j) + L^{ext}(d_j). \end{aligned} \tag{5.11}$$

If there is any *a priori* information about the *code* bits c_m we can include it by forming the log *a priori* probability ratio

$$l(c_m) = \log \frac{Pr(c_m = 1)}{Pr(c_m = 0)} \tag{5.12}$$

and replacing $\mathcal{L}(c_m)$ with $\left[\mathcal{L}(c_m) + l(c_m)\right]$ in (5.7), or equivalently, $\frac{2}{\sigma^2} r_m$ by $\left[\frac{2}{\sigma^2} r_m + l(c_m)\right]$ in (5.9). Also, if there is any *a priori* information about the data bits d_j it can also be included by straightforward addition to $L(d_j)$—this would be $L^{apr}(d_j)$ in (5.2).

5.1.3 LAPP Ratios for Systematic Block Codes with Fixed Coordinates

We can also easily show that for some simple systematic block modulation codes, the expression (5.3) reduces to one which is intuitively satisfying. We consider these simple block codes because we shall later show that their properties greatly enhance their performance on channels like precoded PR channels that are detected with a trellis-based LAPP algorithm like the BCJR or SOVA.

A systematic $(0, k)$ block code of rate $k/(k+1)$ can easily be constructed by simply appending a modulation *one* to each k-bit data block. We refer to this modulation *one* as a fixed coordinate because it is independent of the actual data block. A similar construction using both modulation *zero*s and *one*s can be used to generate systematic block $(d, 2d+1)$ modulation codes with the very low rate of $1/(2d+1)$. For this case the string $0^{(d)}10^{(d)}$ is appended to each data bit.[10] These fixed-coordinate systematic block codes are contrasted with, say, fixed parity codes (e.g. single parity codes (SPCs)—a class of which we discussed in Section 4.3.1 in

[10] Of course, much higher rate fixed-coordinate block (d, k) codes for arbitrary d and k exist. These codes will necessarily have fewer than $2d+1$ fixed coordinates.

which the parity bits are not necessarily identical and depend on the data bits. Clearly, these fixed-coordinate systematic codewords possess precisely the same distance properties as the data blocks (e.g. the full set of κ-tuples for an (n, κ) block code) and therefore, to obtain LAPP ratios for the data bits from soft information for the coded bits, we simply discard the soft information for the inserted parity bits and use the conversion (5.6) on the remaining κ values.

Let $d_E(\boldsymbol{a}, \boldsymbol{b})$ denote the Euclidean distance between vectors $\boldsymbol{a}$ and $\boldsymbol{b}$, and let

$$\mathcal{M}_i \triangleq \exp\left[-\frac{1}{2\sigma^2} d_E^2\left(\boldsymbol{r}, 2\boldsymbol{c}^{(i)} - \mathbf{1}\right)\right].$$

Also, without loss of generality we represent $\boldsymbol{r}$ as the concatenation of the systematic portion $\boldsymbol{r}_s$ and the fixed parity $\boldsymbol{r}_p$, i.e., $\boldsymbol{r} \equiv \boldsymbol{r}_s | \boldsymbol{r}_p$. For a fixed-coordinate systematic block code,

$$d_E^2\left(\boldsymbol{r}, 2\boldsymbol{c}^{(i)} - \mathbf{1}\right) = d_E^2\left(\boldsymbol{r}_s, 2\boldsymbol{d}^{(i)} - \mathbf{1}\right) + C(\boldsymbol{r}_p) \tag{5.13}$$

with $C(\boldsymbol{r}_p)$ being some positive constant which depends only on $\boldsymbol{r}_p$. So we may write

$$L(d_j) = \log\left\{\frac{\sum\limits_{i:d_i^{(j)}=1} \mathcal{M}_i}{\sum\limits_{i:d_i^{(j)}=0} \mathcal{M}_i}\right\} = \log\left\{\frac{\sum\limits_{i:\,d_i^{(j)}=1} \exp\left[-\frac{1}{2\sigma^2} d_E^2\left(\boldsymbol{r}_s, 2\boldsymbol{d}^{(i)} - \mathbf{1}\right)\right]}{\sum\limits_{i:\,d_i^{(j)}=0} \exp\left[-\frac{1}{2\sigma^2} d_E^2\left(\boldsymbol{r}_s, 2\boldsymbol{d}^{(i)} - \mathbf{1}\right)\right]}\right\}. \tag{5.14}$$

For an (n, κ) systematic block code, this results in

$$\begin{aligned} L(d_j) &= \log\left\{\frac{\sum\limits_{i:\,d_i^{(j)}=1} \exp\left[\frac{1}{\sigma^2}\sum\limits_{l=0}^{\kappa-1} r_{s_l}(2d_l^{(i)} - 1)\right]}{\sum\limits_{i:\,d_i^{(j)}=0} \exp\left[\frac{1}{\sigma^2}\sum\limits_{l=0}^{\kappa-1} r_{s_l}(2d_l^{(i)} - 1)\right]}\right\} \\ &= \log\left\{\frac{\exp\left[\frac{r_{s_j}}{\sigma^2}\right] \cdot f(\boldsymbol{r})}{\exp\left[-\frac{r_{s_j}}{\sigma^2}\right] \cdot f(\boldsymbol{r})}\right\} = \frac{2r_{s_j}}{\sigma^2} = \frac{2r_j}{\sigma^2} \end{aligned} \tag{5.15}$$

which is tantamount to discarding the soft parity and applying the conversion (5.6) to the remaining κ values. Thus, the computation gives an intuitively satisfying result for fixed-coordinate systematic block codes.

5.1.4 Block LAPP Algorithm Average Complexity

Using the form (5.7) we can obtain an estimate of the LAPP computational complexity per decoded bit d_j. We assume a general nonsystematic code with *a priori* information on only the $\{c_m\}$. Following [59], we assume $\log(e^x + e^y)$ can be implemented to arbitrary precision with 1 max operation and 1 table lookup.

(i) computation of $\{\mathcal{L}(c_m)\}$ from $\{r_m\}$:

n multiplications per codeword $\Rightarrow n/\kappa$ per decoded bit;

(ii) addition of *a priori* information to $\{\mathcal{L}(c_m)\}$:

n additions per codeword $\Rightarrow n/\kappa$ per decoded bit;

(ii) computation of $\log \sum_{i \in \mathcal{S}_1(j)} \exp\left[\sum_{m:c_m=1} \mathcal{L}(c_m)\right]$:

$2^{\kappa-1} - 1$ max operations per decoded bit;

$(n/2 - 1) \cdot 2^{\kappa-1} + (2^{\kappa-1} - 1)$ additions per decoded bit (have assumed average codeword weight is $n/2$ so that there are on average $n/2 - 1$ additions in the sum $\sum_{m:c_m=1} \mathcal{L}(c_m)$);

$2^{\kappa-1} - 1$ table lookups per decoded bit;

Accounting for the denominator of (5.7) doubles the number of operations in (ii), and requires 1 additional subtraction (which we count as 1 multiplication by -1 and 1 addition) to complete the computation. We summarize these results in the following table;

Table 5.1. Average complexity per decoded bit of evaluating (5.7).

Operation	Block code LAPP
max operations	$2^{\kappa} - 2$
additions	$2^{\kappa-1}n + n/\kappa - 1$
multiplications	$n/\kappa + 1$
lookups	$2^{\kappa} - 2$

If we exclude lookups and assume that the max, addition, and multiplication have equal cost we can express the overall complexity in Table 5.1 for given n and κ, in terms of an equivalent rate $1/2$ binary-input trellis code with memory M as given in [59] for the BCJR algorithm. For example, implementing (5.7) for a rate $4/5$ block code has roughly the complexity—about 56 operations/decoded bit—of operating the BCJR algorithm over a 2-state ($M = 1$) trellis, while for a rate $8/9$ block code, the equivalence is to a 64-state ($M = 6$) trellis (roughly 1300 operations/decoded bit).

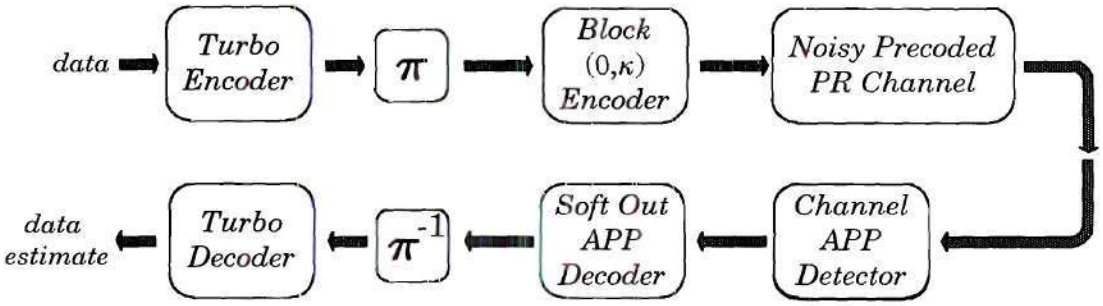
5.2 Standard Concatenation in AWGN Channels—Simulation

Now that we have shown how to obtain soft information from a $(0, k)$-constrained block code we will evaluate, by simulation, the performance of the ('standard concatenation') system of Figure 3.1(a) using some of the $(0, k)$-constrained block codes constructed in Chapter 4. We show how the isolated performance of the codes with an ML (or MAP) decoder in AWGN relates to the performance of the overall system of Figure 3.1(a).

There is an important point to note concerning the interface between the turbo decoder and the LAPP algorithm just described. For an (n, κ) block code, within a particular data block, there is some correlation between the $\{L(d_j)\}_{j=0}^{j=\kappa-1}$ because

they are all computed from the same $\boldsymbol{r}$. The standard implementation of the BCJR algorithm is based on the assumption of independent $L(d_j)$ so an additional interleaver/deinterleaver pair is added to the system of Figure 3.1(a) to remedy this—as shown in Figure 5.1. This interleaver really need not be random or large—a regular block interleaver with a depth of a few $(0, k)$ block-lengths ought to suffice since the memory with which we are concerned extends only over the κ input bits for the (n, κ) $(0, k)$ block code. Removing this additional random interleaver/deinterleaver pair results in a loss of a few tenths of a dB.

(a)



(b)

Figure 5.1. (a) System of Figure 3.1(a) slightly augmented to include an extra interleaver/deinterleaver pair in an AWGN channel; (b) same as (a) but for the noisy precoded PR channel.

As a first example of how the ML (or MAP) decoder performance of the isolated block code relates to the performance of the overall concatenated system, we again compare the performance of the two codes $C_1(\boldsymbol{d})$ and $C_2(\boldsymbol{d})$ from Section 4.1.2 in the configuration of Figure 5.1(a). Figure 5.2 shows the results of this simulation.

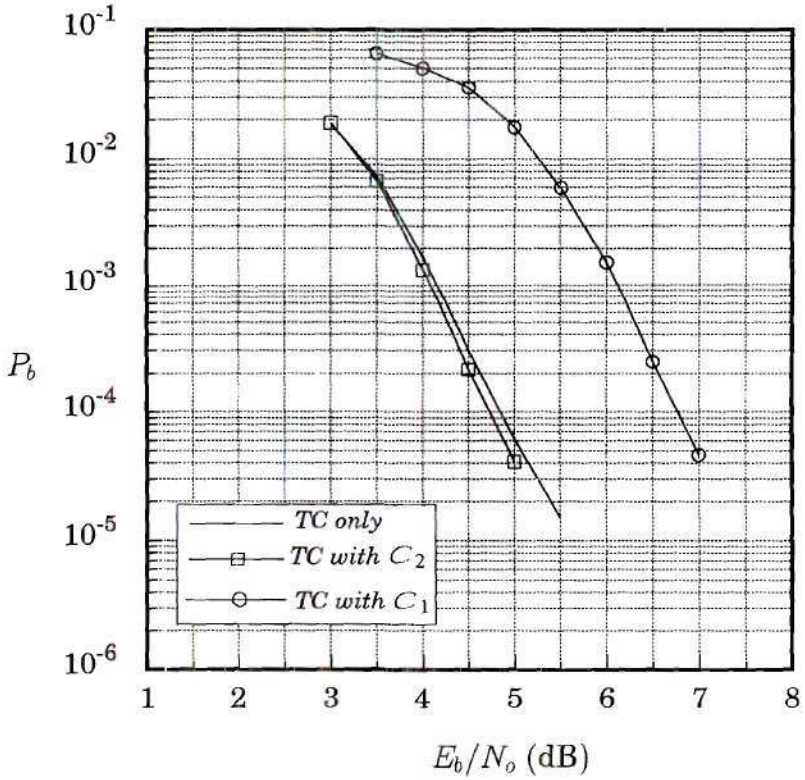


Figure 5.2. BER comparison for system in Figure 5.1(a). Turbo code is rate $8/9$ $(7, 5, 1000)$. 5 turbo decoder iterations.

Observe that the rate loss in Figure 5.2 can be fairly accurately deduced from Figure 4.3 if we compare portions of the two graphs at the same values of E_b/N_o. We observe from Figure 4.3 that, over the range 3 dB $\leq E_b/N_o \leq$ 5 dB (for which Figure 5.2 is plotted), the rate loss in Figs. 4.3 and 5.2 are about equal ($\approx$ 2 dB). The slightly greater rate loss suffered by $\mathcal{C}_1(d)$ in Figure 5.2 is due to error propagation within $\mathcal{C}_1(d)$. Observe that the difference in performance between the two codes in Figure 4.3 reduces with increasing E_b/N_o; therefore the advantage that $\mathcal{C}_2(d)$ has over $\mathcal{C}_1(d)$ is greatest at lower values of E_b/N_o. So we make the observation that the range of operation of the turbo code with which we concatenate the short block code will affect the rate loss experienced by the short block code in the concatenated system. So in a sense, there is benefit from "matching" the turbo code operating range to that of the block constrained code. So in Figure 5.2 $\mathcal{C}_1(d)$ suffers an effective rate loss of almost 2 dB whereas $\mathcal{C}_2(d)$ completely recovers its rate loss. This result shows that in order to avoid severe error propagation from a block $(0,k)$-constrained code that is used in the standard system of Figure 5.1(a) the block code must be carefully designed with very careful attention paid to the inter-codeword Hamming distances and the encoder. The simulations also show that, in exchange for moderate complexity increase, a modest coding gain is available for short high-rate $(0,k)$-constrained codes in the standard system of Figure 5.1(a). Figure 5.3 shows the performance of $\mathcal{C}_5$, $\mathcal{C}_5''$, and $\mathcal{C}_5^{4\prime}$ used as the block $(0,k)$ in Figure 5.1(a). $\mathcal{C}_5$ and $\mathcal{C}_5''$ each provide a modest coding gain of about 0.5 dB and 0.1 dB respectively, while $\mathcal{C}_5^{4\prime}$ is able to recover close to 0.7 dB of its nominal rate loss of 1 dB. Again, note that the relative performance among $\mathcal{C}_5$, $\mathcal{C}_5''$, and $\mathcal{C}_5^{4\prime}$ is consistent, for given E_b/N_o, with that of Figure 4.5.

In [30] it is shown that an outer rate 8/9 SPC provides a net coding gain of 1.5 dB to a rate 1/2 inner convolutional code. A soft-output Viterbi algorithm (SOVA)

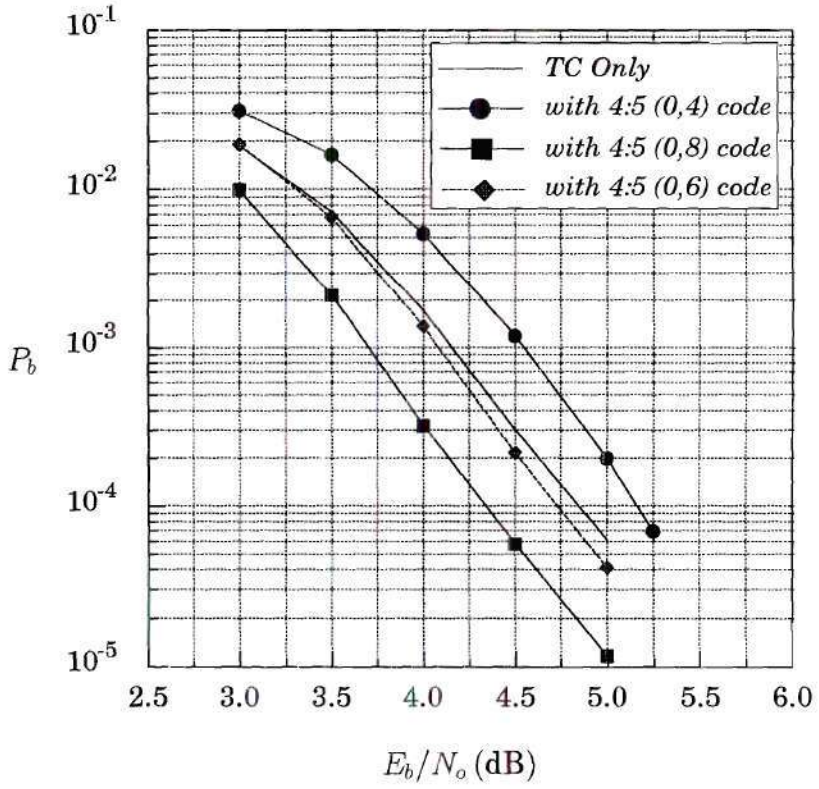


Figure 5.3. The codes $\mathcal{C}_5$, $\mathcal{C}_5''$, and $\mathcal{C}_5^{4\prime}$ used as the block $(0, k)$ in Figure 5.1(b). Turbo code is rate 8/9 $(7, 5, 1000)$. 5 turbo decoder iterations.

first decodes the convolutional code and then the SPC code is used to complement the least reliable bit in the event of a failed parity check. As demonstrated in Section 3.1 it is possible to obtain a $(0, k')$-constrained system by reversing the positions of the turbo code and the $(0, k)$ code—the penalty being that $k' > k$. We have observed from simulation that this constrained-input system is inferior to that of Figure 5.1(a). Using either the block SISO decoder we discussed in Section 5.1 or the method described in [30] (they both give identical performance) we have observed error rate performance for this reversed configuration that is about 1 dB worse than that shown in Figure 5.3 for the codes $\mathcal{C}_n$.

5.3 Standard Concatenation on PR Channels—Simulation

Partial-response channels have become the norm for magnetic data storage channels—currently the most common PR targets are the PR4 (modified duobinary or class IV) with system polynomial $1 - D^2$, and extended PR4 (EPR4) with system polynomial $(1 - D)(1 + D)^2$. Since the PR4 target is equivalent to two independent time-interleaved dicode channels (each with system polynomial $1 - D$) we discuss this channel also in terms of the dicode channel.

Wolf and Ungerboeck [76] have shown that good trellis codes for the dicode channel can be obtained by selecting convolutional codes with maximal free Hamming distance and then using the recursive precoder $(1 \oplus D)^{-1}$ to effectively invert the dicode channel. Therefore, by driving the precoded dicode channel with a code with large Hamming distance, we are guaranteed large Euclidean distance between sequences at the output of the channel. The runlength k constraint is satisfied by using a nontrivial coset of the convolutional code. Uchôa Filho and Herro

[71] have extended this technique to the general class of PR polynomials of the form $(1-D)(1+D)^n$ for which the corresponding precoder is $(1 \oplus D)^{-n-1}$. But Hole and Ytrehus [32] have shown that selecting the convolutional codes that have maximal Hamming free distance does not necessarily yield trellis codes with the largest channel-output squared Euclidean distance. Similarly, within the context of the system of Figure 5.1(b), we argue that, the best performance is not necessarily achieved by selecting $(0,k)$ codes with excellent Hamming distance distribution but the reasons here are somewhat different. So, the approach taken here in applying rate $(n-1)/n$ $(0,k)$ codes to PR channels is to apply the constrained code to a precoded channel. The example we use here is the precoded dicode channel.

When the channel is a precoded PR target, the soft output from the channel can be obtained by applying either the SOVA [30] or BCJR [7] algorithm to the trellis of the precoded PR target—we use the latter. From the remarks made in the preceding paragraph the four code families listed in Table 4.3 will be expected to perform well on the precoded dicode channel due to their very good codeword distance distribution properties. But they are not near-optimal for precoded PR channels, as they are for the AWGN channel in the system of Figure 5.1(a), since it can be demonstrated that at least one other family of rate $(n-1)/n$ $(0,k)$-constrained codes produces channel output sequences with superior inter-codeword Euclidean distance distribution. The systematic modulation code in which the modulation *one* is *appended* to each $(n-1)$-block of data bits; i.e., $\boldsymbol{d} \mapsto \boldsymbol{d}|1 = C_{nsa}(\boldsymbol{d})$ is one such code. While we have not been able to theoretically identify the rate $(n-1)/n$ code family that generates precoded dicode channel output sequences with the optimum inter-codeword Euclidean distance distribution (as was done for the AWGN channel), we show that the simple systematic $(0,n-1)$-constrained code performs very well—better than any of the

codes listed in Table 4.3.[11] This is not only because the systematic code generates channel output sequences with better Euclidean distance distribution; it in fact has more to do with the single fixed coordinate (the modulation *one* that is used to enforce the constraint) that appears in every block of n channel bits. These *one*s represent very strong *a priori* information for the BCJR-based channel APP detector that can be used to great effect in 'holding down' the algorithm after every n epochs and guiding it towards the correct solution. This effect is equivalent to periodically 'pinning' or 'fixing' certain states in the trellis as decoding progresses along the time axis.

Since the input to the precoded PR channel is $(0, k)$-constrained, we could, in principle, employ a Cartesian product-based APP channel detector (following the discussion in Chapter 3). For the relatively large values of k involved, this great additional complexity is prohibitive; additionally, the rewards are insignificant for large k and low η as shown in Section 3.1.4. Therefore, it is better to obtain the LAPP ratios in two independent steps (from the channel APP detector and then the SISO decoder for the block code, respectively, as shown in Figure 5.1(b)). There is therefore a contribution to the performance of the entire system from both the Hamming distance properties of the $(0, k)$ code (used by the SISO block decoder), and, the way in which the $(0, k)$ code interacts with the channel APP detector. We have observed a tradeoff in these two contributions. We have found that the differences alone, in the distribution of Euclidean distances between channel output sequences for different rate $(n-1)/n$ codes (for the same n), do not significantly

---

[11] Placing the modulation *one* at the beginning of the $(n-1)$-block, i.e., $\boldsymbol{d} \mapsto 1|\boldsymbol{d} = \mathcal{C}_{nsp}(\boldsymbol{d})$, is demonstrably inferior to $\mathcal{C}_{nsa}$, $\mathcal{C}_n$, $\mathcal{C}'_n$, $\mathcal{C}''_n$, and $\mathcal{C}^{4'}_n$ in Euclidean distance distribution at the output of the precoded dicode channel.

affect the BER immediately at the output of the channel APP detector—for the precoded dicode case. This has been investigated primarily by computer simulation. We have also verified (using a limited search algorithm) that, for $3 \leq n \leq 10$, no rate $(n-1)/n$ codes result in precoded dicode channel output sequences with minimum squared Euclidean distance greater than 8.[12] So, equivalently, no distance enhancing rate $(n-1)/n$ codes for the precoded dicode channel exist for $3 \leq n \leq 10$. For $n = 2$ though such a code does exist as can be verified from Table 5.2.[13]

Table 5.2. Maximum set size, rate, and minimum k for $(0, k)$ block codes which produce precoded dicode channel output sequences with Euclidean distance > 8 (± 1 input into dicode channel).

n	max. set size	max. block rate	min. k
2	2	1/2	1
3	3	1/3	2
4	6	2/4	3
5	10	3/5	5
6	16	4/6	3
7	31	4/7	2
8	57	5/8	2
9	102	6/9	4
10	198	7/10	4

Therefore, the primary way in which rate $(n-1)/n$ codes affect the channel detector output error rate is by the amount of *a priori* information there is about the coded bits. This is where the tradeoff becomes obvious. The rate $(n-1)/n$ codes possessing the most significant *a priori* information are the single-fixed-coordinate systematic codes; these codes also have the poorest possible Hamming distance distribution of

[12] Assuming ± 1 inputs, the free distance of the dicode channel is 8.

[13] For $n = 2$ this code is the Frequency Modulation code [34].

all rate $(n-1)/n$ codes. Conversely, the rate $(n-1)/n$ codes possessing the best possible Hamming distance distribution (the SPC codes $\mathcal{C}_n$ and $\mathcal{C}_n^*$) have absolutely no *a priori* information (all the code bits are uniformly distributed across the code). So it is reasonable to attempt to determine which property of the codes (distance or *a priori* information) is more valuable in terms of low BER. Simulation results have shown that the latter is the more valuable of the two. In addition, the single-fixed coordinate systematic codes possess a runlength k parameter of $n-1$ which is half of that for $\mathcal{C}_n$ (recall $\mathcal{C}_n$ has $k = 2n-2$).

Figure 5.4 shows the simulated performance of several rate $(n-1)/n$ $(0,k)$ codes used in the system of Figure 5.1(b). It shows the superiority of the systematic codes $\mathcal{C}_{ns}$ (the systematic $\mathcal{C}_{nsa}$ and $\mathcal{C}_{nsp}$ behave identically and so are simply referred to as $\mathcal{C}_{ns}$). In contrast to the case for the AWGN channel, none of the codes is able to completely recover its rate loss but $\mathcal{C}_{ns}$ is able to recover close to half the dB amount, that is, the net rate loss suffered by $\mathcal{C}_{5s}$ is 0.4 dB (versus a nominal 1 dB) and that suffered by $\mathcal{C}_{9s}$ is just under 0.3 dB (versus a nominal 0.5 dB). When $\mathcal{C}_{5s}$ and $\mathcal{C}_{9s}$ are used without using the *a priori* information in the channel APP detector the performance degrades substantially ($\mathcal{C}_{5s}$ loses 0.4 dB of gain and $\mathcal{C}_{9s}$ about 0.2 dB with respect to that shown in Figure 5.4). In addition to its better error rate performance, $\mathcal{C}_{ns}$ benefits from the very simple SISO decoder discussed earlier in Section 5.1.3. Also, the use of the *a priori* information does not increase the complexity of the BCJR algorithm in any significant way. So we observe that the regular spacing of modulation *one*s in the systematic modulation codes can be put to effective use when an APP detector is used to detect a channel. We shall see in the next chapter that a serial concatenated ECC can be designed to exploit this property more fully.

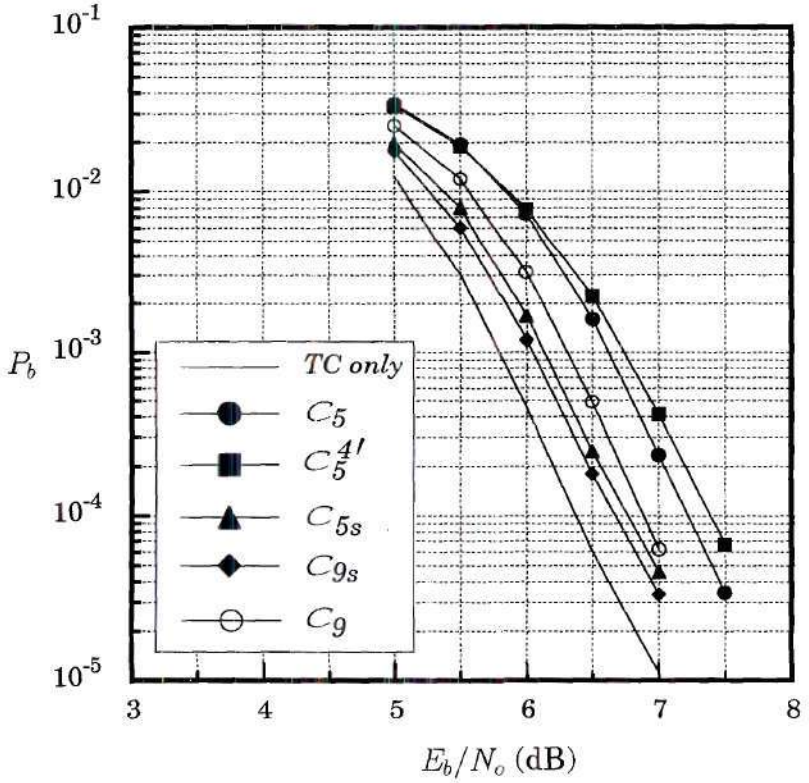


Figure 5.4. BER performance for system of Figure 5.1(b) on a precoded dicode channel. $\mathcal{C}_{5,9}$ are SPC codes of length 5 and 9, respectively, $\mathcal{C}_{5s,9s}$ are systematic modulation codes, and $\mathcal{C}_5^{4'}$ is defined in Section 4.3. Turbo code is rate $8/9$ $(7, 5, 1000)$; 5 iterations.

Since $\mathcal{C}_n$ contains codewords of identical parity, the memory of the two-state precoded dicode channel (see Figure 2.4(b)) effectively extends only over n epochs. The implication of this is that ML decoding may be independently performed on n-blocks of the channel output at a time without loss of performance. This is so because at the end of each encoded n-block the terminal state is the same for all codewords. For example, beginning in state '0', all codewords terminate the trellis in state '1' after the first block; the second n-block terminates the trellis in state '0', the third in state '1' and so on. So successive n-blocks are independent and therefore the general SISO algorithm (5.2) can be used to derive faithful soft outputs from the channel (sampled matched filter output) without having to use either the BCJR or SOVA algorithms.

CHAPTER VI

(d, k)-CONSTRAINED SERIAL CONCATENATION

In this chapter we propose several systems for incorporating (d, k) constraints into interleaved serial concatenated systems. In particular, we show how to very effectively incorporate a systematic $(0, k)$ encoder into interleaved serial concatenated systems. As we discussed in Section 2.2.1, finite-state encoders for (d, k) constraints can be designed with the state-splitting algorithm. We show how to incorporate such finite-state encoders for (d, k) constraints into these serial concatenated systems. As examples, we illustrate the scheme with the rate 1:2 $(1, 3)$ MFM encoder (see Section 2.2) and a rate 2:3 $(1, 7)$ finite-state encoder, [2], [73].

6.1 $(0, k)$-Constrained Serial Concatenation

Recently, Souvignier *et al.* [68] and almost simultaneously McPheters *et al.* [51] have shown that excellent performance is achieved by the simplified interleaved serial concatenated structure (see Figure 2.9) for which the outer encoder is a high-rate (recursive) convolutional encoder and the inner encoder a precoded PR target. For precoded PR targets both papers show that, even though it is much simpler, this simplified structure performs as well as that for which the outer code is a more complex turbo code of equivalent rate and encoder trellis size. The structure

possesses one less interleaver and one less convolutional encoder compared with that using an outer turbo code. We will show how to effectively employ this serial concatenated structure on $(0, k)$-constrained channels. In a general sense, we can observe that the (slightly augmented) standard configuration of Figure 5.1(a) can also be applied when the ECC is a serial concatenated structure—we simply replace the turbo encoder and decoder with those for a serial concatenated code; so all the comments in Section 5.2 and 5.3 hold for this case also. But with the simplified serial concatenation systems for which the inner 'encoder' is a suitably precoded PR target some simplifications are possible.

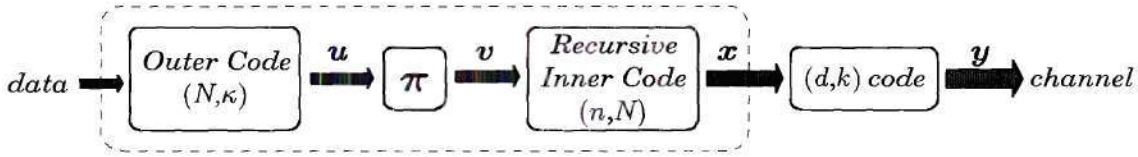
6.1.1 Variations on the $(0, k)$ Encoder Position

For the serial concatenation system of Figure 2.9 there are three possible locations for placement of a $(0, k)$-constrained encoder. These three configurations are shown in Figure 6.1. Each of these positions has different implications for the operation of the full system and the nature of the appropriate constrained code. These concerns are also affected by whether or not the inner encoder is just a recursive precoder or a precoded PR target. Note that when the inner recursive code is rate-1 all three configurations have identical overall rate equal to the product of the rate of the $(0, k)$ code and that of the outer code.

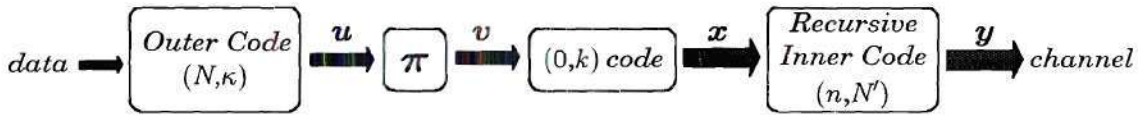
Configuration (a): Configuration (a) is no different from the system discussed in Sects. 5.2 and 5.3 (if we include an extra interleaver/deinterleaver pair as in Figure 5.1) and can accommodate a general (d, k) code whose decoder can deliver faithful soft outputs to the serial concatenated decoder. So all the results in that section for both the AWGN and precoded dicode channel are applicable here as well. Recall that

for precoded PR trellises (using the precoder (2.7)) the channel output sequences possess the same value of runlength parameter k as the precoder input.

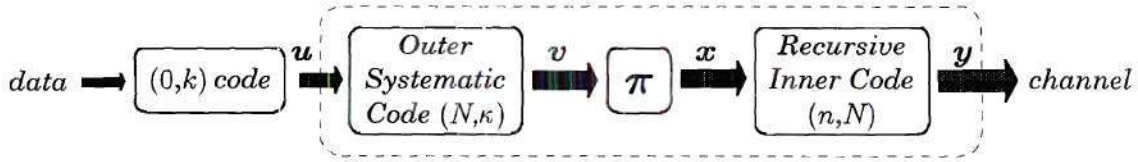
Configuration (b): For configuration (b) there are several concerns for a practical system that do not arise in configuration (a). First, the inner code will affect the structure of the constrained sequence that is delivered to the channel.



(a)



(b)



(c)

Figure 6.1. Different arrangements for $(0, k)$-constrained serial concatenation.

For the system of Figure 6.1(b) we will first comment on the passage of extrinsic information among the decoders and then highlight the results. We assume a block

$(0, k)$ code.

The standard iterative decoding algorithm for interleaved serial concatenation requires that the APP detector operating over the trellis of the recursive inner encoder ('inner APP') compute extrinsic information for $\boldsymbol{x}$ from the noisy received sequence $\boldsymbol{y}$ [64]. So the inner APP computes (using the BCJR algorithm) the extrinsic information

$$L^{ext}(x_j|\boldsymbol{y}) = \log\frac{Pr(x_j = 1|\boldsymbol{y})}{Pr(x_j = 0|\boldsymbol{y})} - L^{apr}(x_j) \tag{6.1}$$

which is the input to the SISO block. $L^{apr}(x_j)$ is the extrinsic information received on the previous iteration—on which we comment in just a moment. $L^{ext}(x_j|\boldsymbol{y})$ is then passed to the SISO block for the $(0, k)$ code which then computes, using (5.7), $L(v_j|\boldsymbol{x})$ which is then deinterleaved and passed to the outer APP as *a priori* information on $\boldsymbol{u}$. To be strict, (5.7) assumes $\boldsymbol{x}$ is soft information (according to our definition at the beginning of Section 5.1) so $\mathcal{L}(c_m)$ should be replaced with $(\sigma^2/2)\mathcal{L}(c_m)$. On the reverse path, the outer APP computes $L^{ext}(u_j)$ which, after interleaving, is used by (5.7) operating in reverse, i.e.,

$$L(c_m) = \log\frac{\sum_{i\in\mathcal{T}_1(m)} \exp\left[\sum_{j:d_j^{(i)}=1} \mathcal{L}(d_m)\right]}{\sum_{i\in\mathcal{T}_0(m)} \exp\left[\sum_{j:d_j^{(i)}=1} \mathcal{L}(d_m)\right]}. \tag{6.2}$$

Using the exact notation associated with (5.2), $\mathcal{T}_1(m)$and $\mathcal{T}_0(m)$ contain the indices i for which $\boldsymbol{d}^{(i)}$ corresponds to a codeword $\boldsymbol{c}^{(i)}$ for which $c_m^{(i)} = 1$, and $c_m^{(i)} = 0$, respect-

ively.[14] (6.2) is used to compute $L(x_j|\boldsymbol{v})$ which is then passed to the inner APP as *a priori* information on $\boldsymbol{x}$, i.e., $L^{apr}(x_j)$. If the block $(0,k)$ code is systematic, then we only need employ (6.2) to compute LAPP ratios for the parity bits (the systematic LAPP values 'pass through' the algorithm unchanged). For an (n,κ) block code with $n \geq \kappa$,

$$\mathrm{E}\Big[d_H\Big(\boldsymbol{c}^{(i)},\boldsymbol{c}^{(j)}\Big)\Big] \geq \mathrm{E}\Big[d_H\Big(\boldsymbol{d}^{(i)},\boldsymbol{d}^{(j)}\Big)\Big], \tag{6.3}$$

and therefore application of (6.2) will ordinarily result in loss of information and a corresponding corruption in the soft information for the code bits c_m. This soft information corruption will be minimized for a systematic code—the systematic bits being immune to information loss—but soft information for parity bits that are functions of the systematic bits will be corrupted. This makes it clear that the best choice of a block encoder in this position, from the viewpoint of minimizing error rate, is the rate $(n-1)/n$ $(0,k)$ systematic encoder. The simulation results to be discussed shortly support this.

If the inner code is a precoded PR channel (this has been shown to be very attractive in terms of simplicity and performance for the precoded dicode channel [51], [68]) then $\boldsymbol{x}$ and $\boldsymbol{y}$ will possess the same $(0,k)$ properties. If, on the other hand, the inner code is of the form $1/(1 \oplus D^N)$, then if $\boldsymbol{x}$ is $(0,k)$-constrained, $\boldsymbol{y}$ will be $(0,k+N)$-constrained for *zeros* and $(0,k')$-constrained for *ones*, with $k' = \min(k,N)$. Note that, just as in Section 3.1, we can employ an APP detector operating over a Cartesian product trellis for the inner code. But for the relatively low efficiencies of systematic $(0,k)$-constrained codes, those results indicate that the Cartesian product

---

[14] For the code in Table 4.1 $\mathcal{T}_1(0) = \{1,3\}$, $\mathcal{T}_0(0) = \{0,2\}$, $\mathcal{T}_1(1) = \{0,1,2\}$, $\mathcal{T}_0(1) = \{3\}$, $\mathcal{T}_1(2) = \{0,1\}$, and $\mathcal{T}_0(2) = \{2,3\}$.

APP detector would actually degrade the error-rate performance. Another advantage of systematic $(0, k)$-constrained codes that we have seen from Section 5.3 is that the APP detector for the inner code can additionally make effective use of the strong *a priori* information represented by the modulation *ones*.

We have simulated the performance of this configuration for the case where the inner code is the precoded dicode channel. For the simulation we have set the large *a priori* values that are theoretically equal to $+\infty$, to whatever large value is necessary to prevent numerical overflow. Figure 6.2 shows that rate $4/5$ $(0, 4)$ and rate $8/9$ $(0, 8)$ constraints can be used with only about 0.15 dB net rate loss. This is equivalent to rate loss recovery of 0.85 dB and 0.35 dB, respectively, for the rate $4/5$ and $8/9$ codes. Although not shown, the performance for $k \in \{5, 6, 7\}$ is coincident with that shown for $k \in \{4, 8\}$ in Figure 6.2. The rate loss recovery does not depend significantly on k because there exists an almost balanced tradeoff between the effectiveness of the *a priori* information in guiding the BCJR algorithm (i.e., the frequency of state pinning), and the rate of the constrained code. When k is small, the progression of the algorithm is tightly controlled by very frequent state pinning. In this case the algorithm is very reliable because the large *a priori* values 'guide' it in the correct direction very frequently. But the effectiveness of all this redundancy is offset by the low constrained code rate. For larger values of k when there is not as much redundancy, the algorithm is not guided as often and is less reliable but the constrained code rate is higher, making up for this. So the two effects almost compensate for each other so that the performance of the full system is only mildly dependent on k.

We notice that this system, with the same systematic $(0, k)$ block codes, outperforms that of Figure 5.1(b) in terms of effective rate loss. For that turbo-coded system, the $8/9$ systematic code suffers an effective rate loss of 0.3 dB whereas the

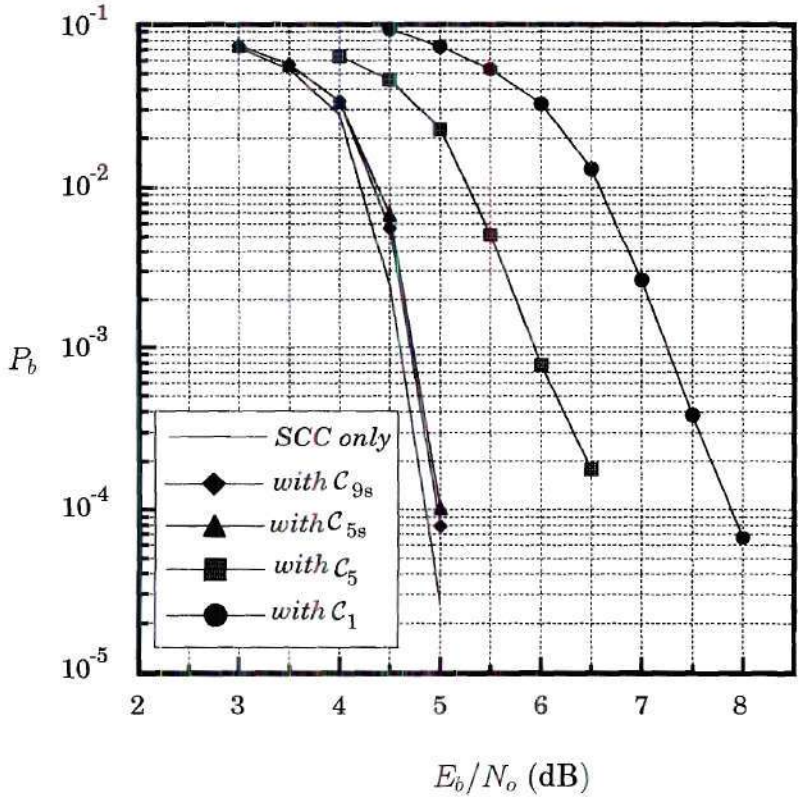


Figure 6.2. Error rate performance of Figure 6.1(b) with several systematic rate $k/(k+1)$ $(0,k)$ codes. C_1 is defined in Table 4.2. Baseline serial concatenated system is a rate 8/9 punctured $(31,33)$ outer RSC encoder, and an inner $1/(1 \oplus D)$-precoded dicode channel. Input block size is 4000.

same code suffers only 0.1 dB in this serial-concatenated system. Also shown in Figure 6.2 are the performance curves for $\mathcal{C}_1$ from Table 4.2 and $\mathcal{C}_5$; the superiority of the systematic modulation codes is again evident. Observe that $\mathcal{C}_5$, because of the dependence of its single parity bit on the systematic bits, suffers greater rate loss in this system than it does in the system of Figure 5.1.

In hard-decision decoding of codes, fixed coordinates (as first defined in Section 5.1.3) in a code do not contribute to the error correction capability of the code and are thus truly overhead. But as we have seen this is not the case when the code in question drives a system with memory (another trellis code or convolutional code, or, an ISI channel). Under these circumstances these fixed coordinates can contribute towards an improvement in the error-rate performance of the larger concatenated system by the mechanism of 'state-pinning' in an APP algorithm like the BCJR. We have also pointed out how they also minimize corruption of information in connection with (6.2).

Configuration (c): Due to the fact that the $(0, k)$-constrained code is located before the random interleaver, configuration (c) is practically only possible with a $(0, k)$-constrained code with fixed coordinates since it can be shown that a subset of all fixed random interleavers can support such a block code. We will show that this configuration outperforms both configurations (a) and (b) when systematic $(0, k)$ codes are used.

First, observe that for this configuration we have the opportunity to exploit the fixed coordinates in *both* of the APP detectors — if the fixed random interleaver can be designed to cooperate with the $(0, k')$-constrained sequence $\boldsymbol{v}$ and produce a $(0, k')$-constrained $\boldsymbol{x}$ as well. Since $\boldsymbol{u}$ is $(0, k)$-constrained it resembles

$$\cdots, 1, \underbrace{\square, \square, \cdots, \square}_{k}, 1, \underbrace{\square, \square, \cdots, \square}_{k}, 1, \cdots \tag{6.4}$$

with the '□'s occupied by user data. Let us index the sequence $\boldsymbol{u}$ as $\{u_i\}_{i=0}^{\kappa-1}$ for a length-κ input sequence $\boldsymbol{u}$ into the outer encoder. Then the indices of the '1's in this constrained sequence $\boldsymbol{u}$ are given by the index sequence

$$\boldsymbol{i} = \{i_j\} = \{k, 2k+1, 3k+2, \cdots, jk+(j-1), \cdots, i_{max}\} \qquad j \in \mathbb{Z}^+ \tag{6.5}$$

and the outer APP detector would make use of these indices. Next, let us assume the outer systematic encoder has rate $(n-1)/n$. This encoder inserts one parity bit after each $(n-1)$ constrained bits and so u_h is translated into $v_{h'}$, that is

$$h \mapsto h' : h' = h + \left\lfloor \frac{h}{n-1} \right\rfloor. \tag{6.6}$$

Therefore the indices of the '1's in the sequence $\boldsymbol{v}$ are now given by the index sequence

$$\boldsymbol{i}' = \{i'_j\} = \left\{ jk + (j-1) + \left\lfloor \frac{jk+(j-1)}{n-1} \right\rfloor \right\}_{j=1}^{j=j_{max}} \tag{6.7}$$

with

$$j_{max} = \max_{i'_z \leq N-1} (z \in \mathbb{Z}^+)$$

As an example, for $k = 4$ and $n = 9$ the '1's in the sequence $\boldsymbol{v}$ at the interleaver input occur at the indices $\boldsymbol{i}' = [4, 10, 15, 21, 27, 32, 38, 43, 49, 55, \cdots]$. Notice that, in general, the spacing between '1's is not uniform, and the encoder output is now $(0, k')$-constrained with k' given by (3.1). We refer to these indices as the 'fixed points' of the sequence.

Let $\boldsymbol{\pi}$ be the set of all the fixed random interleavers π of length N; obviously $|\boldsymbol{\pi}| = N!$. There is a proper subset $\boldsymbol{\pi}^f(k,n) \subset \boldsymbol{\pi}$ containing permutations π_m^f of the

set $\{0, 1, \cdots, N-1\}$ for which all the fixed points are mapped only onto other fixed points. More concisely, for the m^{th} permutation in $\boldsymbol{\pi}^f(k, n)$

$$\pi_m^f(v_{i_g'}) = (v_{i_h'}) \quad g, h, m \in \mathbb{Z}^+ \tag{6.8}$$

We can easily compute the fraction of all permutations of the set $\{0, 1, \cdots, N-1\}$ which are fixed-point with parameters n and k. Let there be j_{max} fixed points in $\{0, 1, \cdots, N-1\}$; there are $j_{max}!$ permutations of these fixed points and obviously $(N - j_{max})!$ permutations of the 'free' points, giving a total of $(N - j_{max})!j_{max}!$ possible fixed-point permutations. Therefore we see that the ratio of possible fixed-point permutations to the total number of permutations is

$$\frac{|\boldsymbol{\pi}^f(k, n)|}{|\boldsymbol{\pi}|} = \frac{(N - j_{max})!j_{max}!}{N!} = \frac{1}{\binom{N}{j_{max}}} \tag{6.9}$$

which makes it easy to see that this is a rather small fraction indeed. This greatly increases the chance of picking a bad interleaver and many more attempts than usual may be required in order to randomly select a good one.

Constructing such a fixed-point interleaver is conceptually fairly straightforward and tacitly given in the foregoing discussion. One simply partitions the integers $\{0, 1, \cdots, N-1\}$ into two sets $\boldsymbol{f}$ and $\boldsymbol{r}$. $\boldsymbol{f}$ and $\boldsymbol{r}$ are each randomized to give $\boldsymbol{f}^\pi$ and $\boldsymbol{r}^\pi$, respectively, and then the $\{f_i^\pi\}$ are inserted into the $\{r_i^\pi\}$ at the indices $\{f_i\}$. For example, with $N = 10$ we might have

$$\boldsymbol{f} \cup \boldsymbol{r} = \{0, 1, \underline{2}, 3, \underline{4}, 5, 6, \underline{7}, 8, \underline{9}\}$$

with the fixed points underlined. So $\boldsymbol{f} = \{2, 4, 7, 9\}$ and perhaps $\boldsymbol{f}^\pi = \{9, 7, 2, 4\}$; also, $\boldsymbol{r} = \{0, 1, 3, 5, 6, 8\}$ with perhaps $\boldsymbol{r}^\pi = \{5, 1, 8, 0, 6, 3\}$. So we obtain the fixed point interleaver as $\{5, 1, 9, 8, 7, 0, 6, 2, 3, 4\}$. So with such an interleaver, the input to the inner encoder will also be $(0, k')$-constrained, and we have already discussed (in

connection with configuration (b)) the implications of this on the eventual runlength parameter for sequences written to the channel.

Next we address the actual decoding algorithm. Since the input to both APP decoders have '1's at fixed positions (different, of course, for each decoder), both decoders can make use of very strong *a priori* values for these indices. So the standard serial decoding algorithm is modified according to these rules.

(i) APP$^{(i)}$ (the decoder for the inner code) computes LAPP ratios $L(\boldsymbol{x})$ for the sequence $\boldsymbol{x}$ using large *a priori* ($= +\infty$) values for the appropriate fixed indices of $\boldsymbol{x}$. Before passing $L(\boldsymbol{x})$ through the deinterleaver, the computed values $L(x_i)$ for $i \in \{\text{inner fixed points}\}$ are ignored and replaced with ∞ since that is what they are known to be.

(ii) APP$^{(o)}$ (the decoder for the outer code) computes LAPP ratios $L(\boldsymbol{u})$ and $L(\boldsymbol{v})$ for the sequences $\boldsymbol{u}$ and $\boldsymbol{v}$, respectively, using large *a priori* ($= +\infty$) values for the appropriate fixed indices of $\boldsymbol{u}$. Before passing $L(\boldsymbol{v})$ through the interleaver, the computed values $L(v_i)$ for $i \in \{\text{fixed points of } \boldsymbol{v}\}$ are ignored and replaced with $+\infty$ since that is what they are known to be.

When decisions on the data are ready to be made, $\boldsymbol{u}$ is simply punctured and sliced. We have simulated this system and modified algorithm using a rate $8/9$ $(31, 33)$ outer RSC encoder and a precoded dicode inner code for several values of k at an interleaver size $N = 4000$. In making comparisons to the baseline (unconstrained) system we have been careful to ensure a constant interleaver size by adjusting the data block size. We show the error rate performance in Figure 6.3. It shows that, for $k = 4$, a net coding gain of 0.1 dB is actually achieved — and for $k = 3$ and $k > 4$ the constrained code can be used essentially without rate loss penalty. The practical

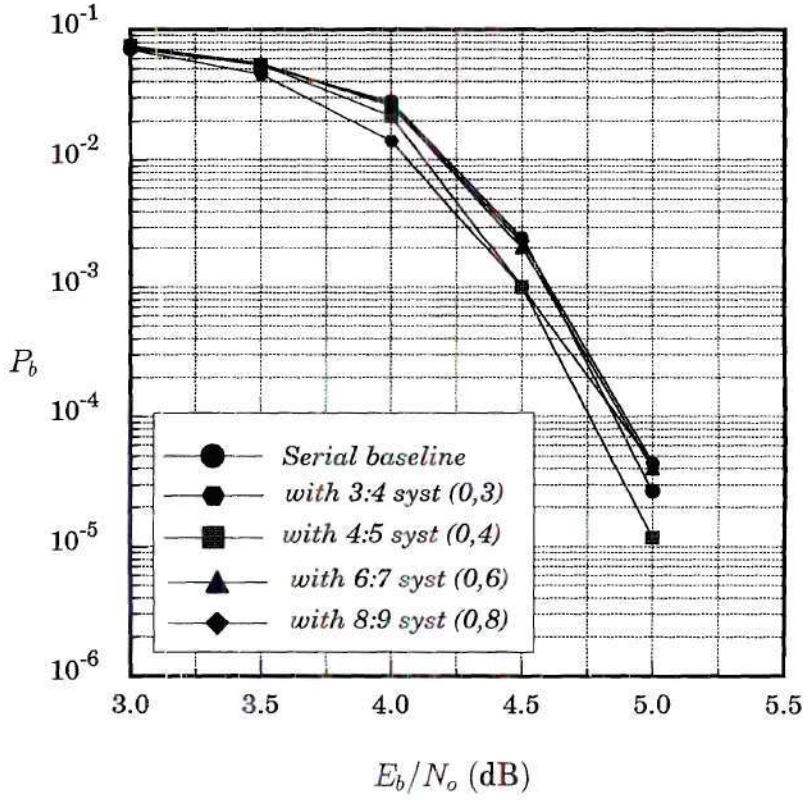


Figure 6.3. Error rate performance for several systematic $(0, k)$ codes in the serial concatenated system of Figure 6.1(c). Baseline unconstrained system has a rate $8/9$ $(31, 33)$ outer RSC encoder, and a precoded dicode inner code. All interleavers are same size ($N = 4504$).

implementation of this system is also not any more complex than that for Figure 6.1(b) except that some extra effort is required for the design of the interleaver.

6.1.2 Comparison to a Competing System

The clever system in Figure 6.4 has recently been suggested by Ryan [62] for implementing $(0, k)$-constrained coding with serial concatenation. It is the only other system for implementing $(0, k)$-constrained coding in the presence of interleaved serial concatenation of which we are aware.

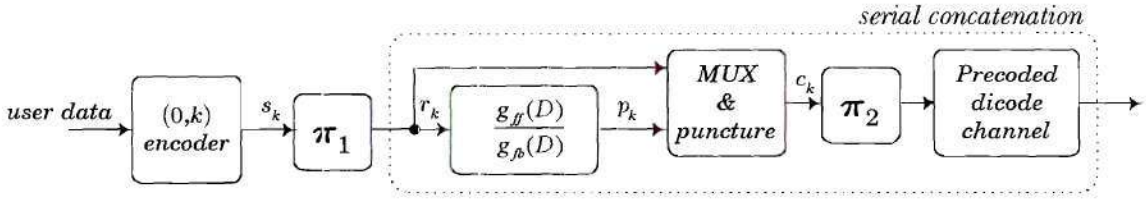


Figure 6.4. "Preinterleaved" $(0, k)$ serial concatenation. [62]

The $(0, k)$ encoder is a general high-rate encoder, not necessarily systematic. The interleaver π_1 is a random interleaver and therefore undoes the constraint in $\boldsymbol{s}$, and so the input to π_2 is unconstrained. But since the outer encoder is systematic π_2 can be designed to restore the original order of the s_k within $\pi_2(\boldsymbol{c})$—that is $\pi_2(\boldsymbol{c})$ will be $(0, k')$-constrained with k' again given by (3.1) when the outer encoder is rate $(n-1)/n$. We clarify the design of the interleaver with an example. Let the sequence

$$\boldsymbol{s} = \{a, b, c, d, e, f, g, h, i, j\}$$

be $(0, k)$-constrained of length 10. $\boldsymbol{\pi_1}$ can be any permutation of $\{0, 1, \cdots, 9\}$. So perhaps $\boldsymbol{\pi_1} = \{4, 7, 1, 9, 5, 8, 0, 3, 6, 2\}$ and therefore $\boldsymbol{\pi_1^{-1}} = \{6, 2, 9, 7, 0, 4, 8, 1, 5, 3\}$. So,

$$\boldsymbol{r} = \pi_1(\boldsymbol{s}) = \{e, h, b, j, f, i, a, d, g, c\}.$$

Let $n = 4$; then $\boldsymbol{c} = \{e, h, b, p_0, j, f, i, p_1, a, d, g, p_2, c\}$, with the three inserted parity bits labeled p_0, p_1, and p_2. Let $\boldsymbol{\pi_2} = \{\pi_{2_0}, \pi_{2_1}, \cdots, \pi_{2_{N-1}}\}$ and let $\mathfrak{v} = \{\mathfrak{v}_0, \mathfrak{v}_1, \cdots\} \subset \boldsymbol{\pi_2}$ be the indices of $\boldsymbol{\pi_2}$ corresponding to the systematic bits. If we arrange matters so that

$$\mathfrak{v}_i = \left(\pi_1^{-1}\right)_i + \left\lfloor \frac{\left(\pi_1^{-1}\right)_i}{n-1} \right\rfloor$$

then the original order of the sequence $\boldsymbol{s}$ will be restored at the output of the interleaver π_2. The remaining indices of $\boldsymbol{\pi_2}$ which correspond to the inserted parity are unconstrained and can be any random permutation of those indices. So continuing with the example,

$$\mathfrak{v} = \{8, 2, 12, 9, 0, 5, 10, 1, 6, 4\}, \text{ and}$$
$$\mathfrak{p}^{\pi} = \boldsymbol{\pi_2} \backslash \mathfrak{v} = \{11, 3, 7\}$$

with $\mathfrak{p}^{\pi}$ being a random permutation of the indices $\mathfrak{p} = \{3, 7, 11\}$ of the parity bits in $\boldsymbol{c}$ ($\mathfrak{p}$ may be obtained by applying (6.6)). So

$$\boldsymbol{\pi_2} = \{8, 2, 12, 11, 9, 0, 5, 3, 10, 1, 6, 7, 4\}$$

and therefore

$$\boldsymbol{\pi_2}(\boldsymbol{c}) = \{a, b, c, p_2, d, e, f, p_0, g, h, i, p_1, j\}$$

which is $(0, k')$-constrained.

This scheme has the advantage of allowing the use of an arbitrary $(0, k)$-constrained code, so a high-rate code can be chosen to minimize rate loss. For example, the nominal rate loss for rates $8/9$ and $16/17$ $(0, 3)$ and $(0, 4)$ codes ([21], [24], [46]) are 0.5 dB and 0.26 dB, respectively. In general, the rate loss will be higher with this scheme than for our proposed system, but the advantage here is that the runlength k parameter can be much smaller.

The performance of this scheme might be improved by employing a Cartesian product-based APP detector for the inner encoder since its input is $(0, k')$-constrained. As we have seen from Section 3.1 this is only attractive for $k' \leq 4$.

6.2 Precoded (d, k)-Constrained Serial Concatenation

The (d, k) constraints with $d > 0$ often find use in controlling ISI on constrained AWGN channels; they have also been used in conjunction with some EPR4 and E^{2}PR4 channels. In particular, the $d = 1$ constraint, when passed through the $(1 \oplus D)^{-1}$precoder, is known to improve the error rate at the output of EPR4 and E^{2}PR4 channels by reducing the number of closed error events at $d^2_{E,min}$—for EPR4—and, by raising $d^2_{E,min}$to the matched filter bound—for E^{2}PR4 [34], [38], [39]. Quite a few well-known encoders for (d, k) constraints have been designed using the state-splitting algorithm and are thus finite-state encoders (i.e. trellis encoders) of rate $p : q$. In this section, we will focus on delivering a (d, k)-constrained sequence to an AWGN channel within an interleaved serial concatenation framework.

(d, k)-constrained sequences from finite-state encoders typically have $d_{min} = 1$ since the FSTD that describes these constraints generates sequences with $d_{min} = 1$. The constrained codes generated by these encoders thus have poor Hamming distance properties and with their accompanying sliding block decoders, the nominal rate loss for the code is assured. In an interleaved serial concatenated system, it is known that a recursive inner encoder is required for interleaver gain. Many finite-state (d, k) encoders are not recursive but they can still be used as the inner encoder in a serial concatenated system if they can be suitably precoded. So we again contemplate the system of Figure 6.1(a) with the "Recursive Inner Code" block

representing a rate-1 precoder, and the "(d, k) code" block implemented as finite-state encoder. We first describe the general method of performing the precoding, and then illustrate the system with two examples.

6.2.1 Precoding a Rate $p : q$ Trellis

We consider a trellis, such as that for a finite-state encoder, whose edges have input labels which are blocks of p bits and output labels which are blocks of q bits. We can contemplate a recursive precoder that operates over the binary extension field GF(2^p). The precoder thus maps each sequence over GF(2^p) to another over GF(2^p); let this mapping be denoted $\mathcal{P}$. The finite-state (d, k) encoder can also be envisioned to map each unconstrained sequence over GF(2^p) to a constrained sequence over GF(2^q). We denote this mapping by $\mathcal{D}$. This description of a precoder structure provides the necessary interface to the rate $p : q$ finite-state (d, k) encoder and ensures that the Cartesian product $\mathcal{P} \times \mathcal{D}$ is well defined. The trellis over which the BCJR algorithm (or equivalent) is operated in the receiver is thus that of the Cartesian product $\mathcal{P} \times \mathcal{D}$. As an example, Figure 6.5 depicts the simplest possible rate-1 recursive precoder—equivalent to a generalized transfer function of $\frac{1}{1+D}$ (i.e., NRZI), with "+" understood to represent the addition operation in GF(2^p) and "D" understood to denote delays in time units of GF(2^p) symbols. When $p = 1$, this structure becomes the familiar binary NRZI precoder. For construction of GF(4) we have used the primitive polynomial $p(x) = 1 + x + x^2$. With α a root of $p(x)$ and primitive, we list the elements of GF(4) as $\{0, 1, \alpha, \alpha^2\}$. We have then assigned each element of GF(4) to a binary two-tuple using $[\alpha, 1]$ as a basis for a vector space construction [74]. The trellis for the precoder in Figure 6.5 is given in Figure 6.6 for GF(4), i.e., $p = 2$.

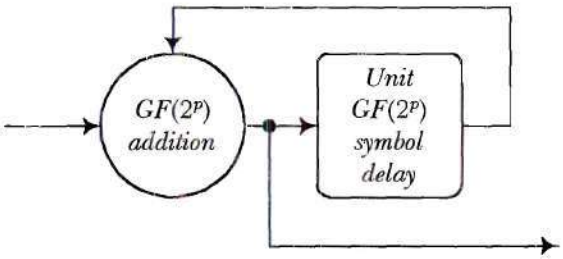


Figure 6.5. Generalized NRZI precoder.

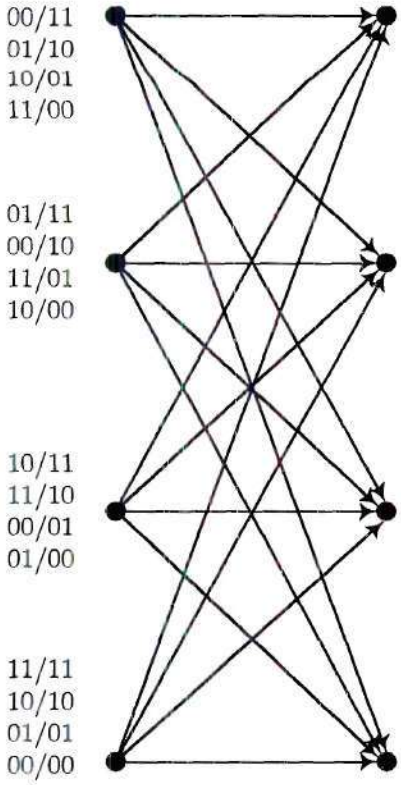


Figure 6.6. $1/(1 + D)$ trellis over GF(4).

Appealing to the fact that GF(2^p) can be viewed as a vector space over GF(2), it is easily appreciated that, in general, with a memory-M precoder over GF(2^p), the

2^{pM}-state trellis with a total of $2^{p(M+1)}$ edges per trellis section is equivalent to a p-fold interleave of the binary 2^M-state trellis with 2^{M+1}edges. The equivalent p-fold interleaved binary version for the precoder in Figure 6.5 would be implemented as is shown in Figure 6.7. This equivalence offers a substantial reduction in implementation complexity when p is large.

Extending nonbinary finite field finite state machines to other familiar transfer functions (and recursive precoders in particular) is straightforward. So, in principle, the binary input to the precoder is parsed into symbols from GF(2^p) and then the precoder transfer function is then implemented at the symbol level using GF(2^p) arithmetic; in practice, the p-fold interleaved binary implementation would be simpler. We next illustrate this precoded serial concatenated system with two examples.

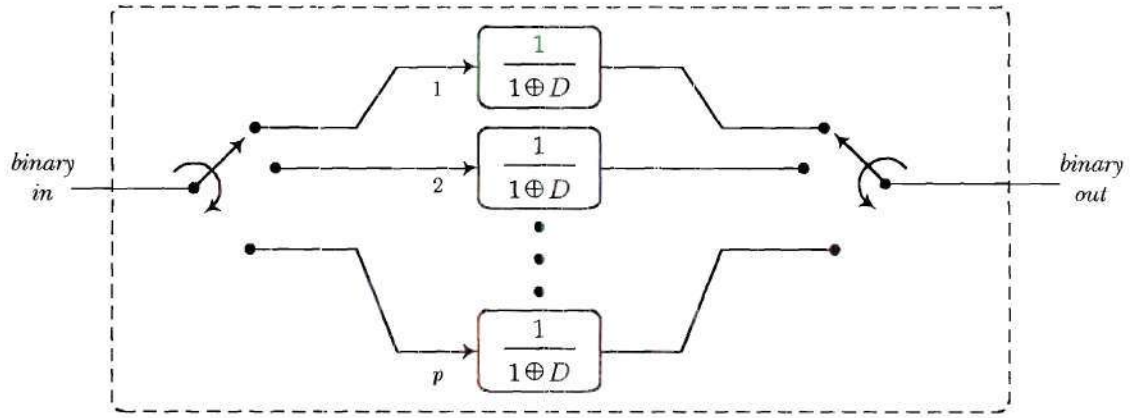


Figure 6.7. p-fold interleaved binary $(1 \oplus D)^{-1}$ precoders. Equivalent to a single $(1 + D)^{-1}$ precoder over GF(2^p).

6.2.2 Example I—Precoded MFM

The encoder for the rate $1:2$ $(1,3)$ MFM code is shown in Figure 2.6; the corresponding trellis is shown in Figure 6.8(a). The nominal rate loss for this systematic code is 3 dB if the parity bits are simply discarded. But use of an optimal MAP algorithm operating on the trellis can recover about 1 dB of this rate loss.

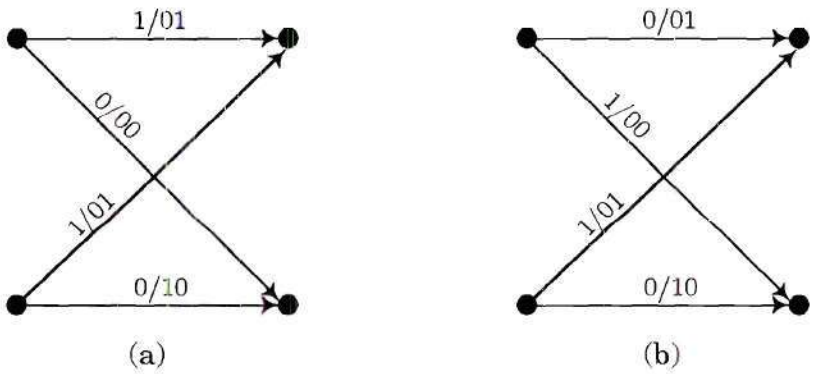


Figure 6.8. (a) MFM trellis; (b) NRZI-precoded MFM trellis.

It is not difficult to see that using this encoder as an inner code in a serial concatenated scheme will not provide any interleaver gain. This is because the encoder is not recursive—a weight-one input error sequence of $[\cdots, 0, 0, 1, 0, 0, \cdots]$ generates the minimum distance error event, also of weight one. A weight-one input error sequence cannot be 'broken up' by a random interleaver.

This heuristic statement can be supported by studying the transfer function of the trellis. Since the trellis describes a nonlinear code, the average transfer function can be derived from the error state diagram of the trellis using the techniques in [14], [51], and [72]. In Figure 6.9 the initial state in the diagram corresponds to the beginning of an error event; the middle state corresponds to a state in which the error event continues (paths do not merge), and the terminal state corresponds to

the termination of the error event. The edge labels on the diagram have the following interpretation: the exponent of D gives the output Hamming weight corresponding to a particular branch of the pairwise error event, the exponent of L gives the length (in trellis sections) of that branch of the pairwise error event, and the exponent of I gives the Hamming weight of the input error corresponding to that branch. The polynomial branch labels have coefficients that reflect an average value for each pairwise contribution. These averages have been taken over all possible pairwise branches for that part of the error state diagram.

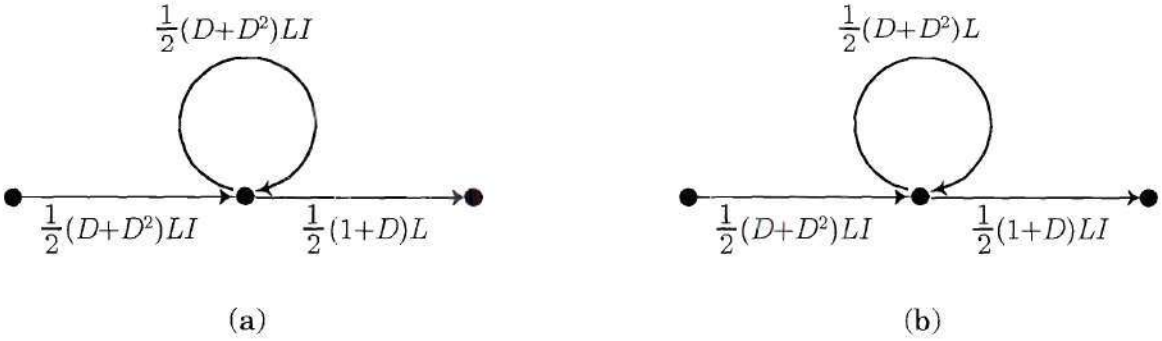


Figure 6.9. Average error state diagrams (a) MFM, (b) NRZI precoded MFM.

The average transfer function for the MFM trellis is easily obtained as

$$\begin{aligned} T_{avg}^{MFM}(D,L,I) &= \frac{\frac{1}{4}(D+D^2)(1+D)L^2I}{1-\frac{1}{2}(D+D^2)LI} \\ &= \frac{1}{4}D(1+D)^2L^2I + \frac{1}{8}D^2(1+D)^3L^3I^2 + \cdots \\ &\quad + \frac{1}{2^{k+1}}D^k(1+D)^{k+1}L^{k+1}I^k + \cdots \end{aligned} \tag{6.10}$$

and that for the NRZI precoded MFM trellis is similarly obtained as

$$T_{avg}^{PMFM}(D,L,I) = \frac{\frac{1}{4}D(1+D)^2L^2I^2}{1-\frac{1}{2}(D+D^2)L} \tag{6.11}$$
$$= \frac{1}{4}D(1+D)^2L^2I^2 + \frac{1}{8}D^2(1+D)^3L^3I^2 + \cdots$$
$$+ \frac{1}{2^{k+1}}D^k(1+D)^{k+1}L^{k+1}I^2 + \cdots$$

It is now possible to see more clearly that, for the MFM trellis, the error event at the free distance (= 1) of the trellis is caused by a single weight-one input error sequence of length two. It can also be seen that there are only two input error sequences that cause error events at $d_H = 2$; one is of input weight one and length two, and the other of input weight two and length three. Therefore a random interleaver placed at the input to this trellis cannot reduce the multiplicities of these most significant error events, that is, interleaver gain is not available.

For the NRZI-precoded MFM trellis on the other hand, we observe that the error event at the free distance is this time caused by a single weight-two input error sequence of length two. In fact, *all* error events are caused by input error sequences of weight two. In addition, as the length of a weight-two input error sequence increases, it becomes associated with an error event at increasing output Hamming distance; the average number of such error events decreases exponentially with increasing error event length. So a random interleaver will function to reduce the effective multiplicity of error events at the free distance of the trellis—thus interleaver gain is available. Due to the low free distance of the trellis, though, a pronounced error floor will be visible in the BER curve for a serial concatenated system with this encoder as the inner encoder. Figure 6.10 shows the simulated error rate performance of this serial concatenated (d,k) scheme in AWGN.

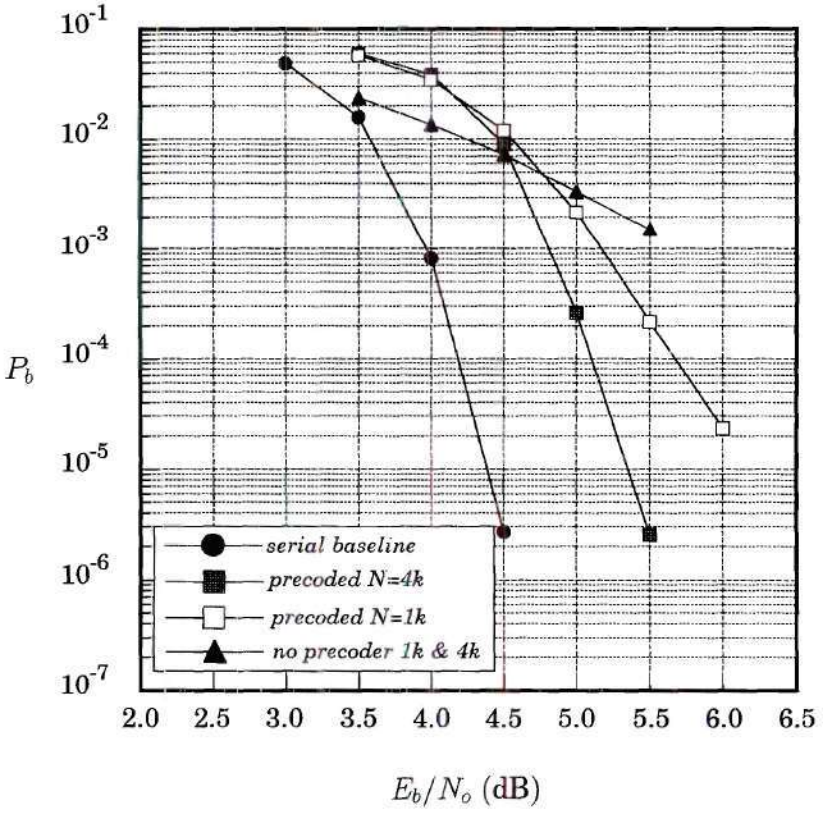


Figure 6.10. Serial concatenation of an outer rate $8/9$ $(31, 33)$ RSC and inner MFM encoder in AWGN. Interleaver size is 4000. 5 iterations. "serial baseline" indicates inner $1/(1 \oplus D)$ precoder only with size-4000 interleaver.

The simulation results confirm that, without the precoder, interleaver gain is unavailable, and the system almost certainly suffers the nominal rate loss of 3 dB—perhaps more. But with the $(1 \oplus D)^{-1}$ precoder, 2 dB of the rate loss is recovered. This rate loss recovery is essentially without additional cost since the precoded MFM trellis has the same complexity as the MFM trellis.

6.2.3 Example II—A Precoded Rate 2/3 (1, 7) Encoder

There are two well-known rate $2/3$ encoders for the $(1, 7)$ constraint. The earlier of the two is due to Adler *et al.* ("AHM") [2] and is a five-state encoder. Weathers and Wolf ("W&W") subsequently showed a four-state version in [73]; four is the minimal number of states for an encoder with these parameters. The trellis for the four-state encoder is shown in Figure 6.11(a). When this trellis is precoded with the GF(4)-extended $(1 + D)^{-1}$a five-state Cartesian product results which is shown in Figure 6.11(b). When the five-state AHM encoder is similarly precoded a six-state trellis results. The nominal rate loss for a rate $2/3$ code is 1.8 dB.

The average transfer function of these nonlinear trellises can, in principle, be derived from the corresponding error state diagrams just as was done for the MFM encoder but due to the number of edges in the trellis this is a laborious task. So the transfer function analysis for this case is omitted. Nevertheless, it can be seen that for the non-precoded four-state trellis $d_{free} = 1$, and at least one error event at this distance is associated with an input error sequence of length two bits (one trellis section) and of weight one. This corresponds to the two parallel edges between the bottom two states (states 0 and 1). This implies that full interleaver gain is unavailable. But since at least one other pairwise error event at d_{free} corresponds to a weight-two input error sequence (compare state transitions $0 \rightarrow 1 \rightarrow 3$ and

$0 \rightarrow 2 \rightarrow 3$), it can be inferred that slight interleaver gain will be observed. Figure 6.12 shows the simulated performance of this system with and without the precoder at two different block sizes for the W&W encoder while Figure 6.13 compares the performance of the two encoders at an interleaver size of 4000. For both encoders, slight interleaver gain is evident for the non-precoded case whereas the more pronounced interleaver gain is visible in the performance of the precoded system. These simulation results show that these precoded systems can be used to apply either of these rate $2/3$ $(1,7)$ encoders with an effective rate loss of under 1 dB—a substantial rate loss recovery of over 0.8 dB compared to the nominal 1.8 dB.

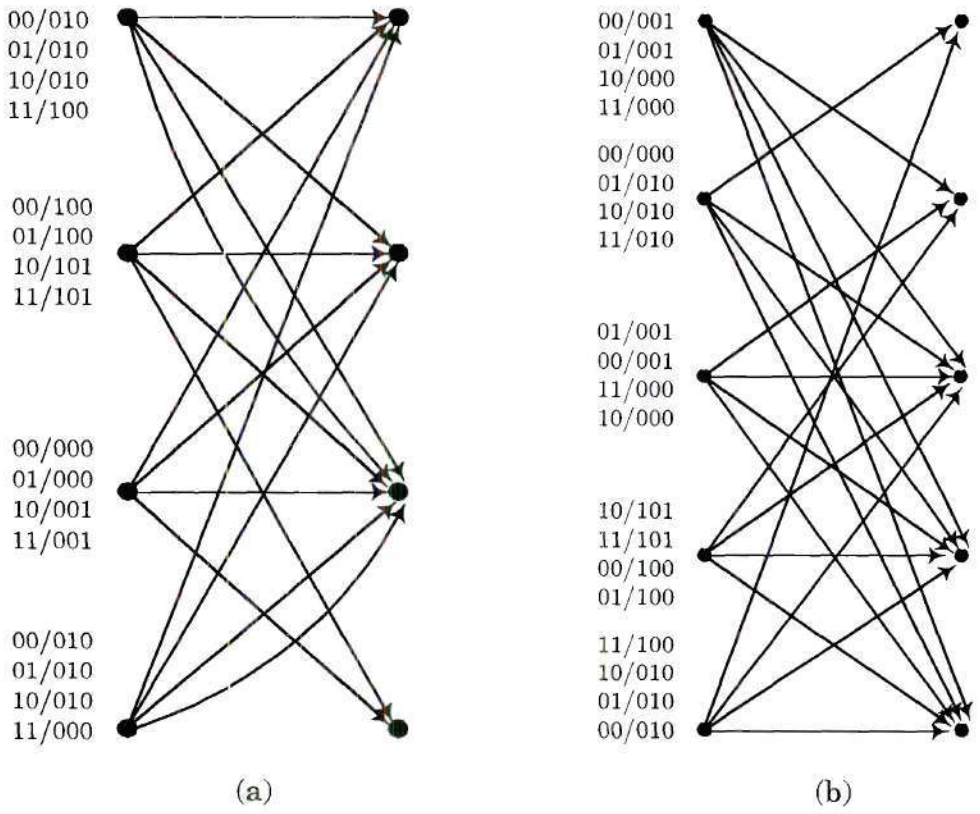


Figure 6.11. (a) Trellis for rate $2/3$ $(1,7)$ W&W encoder. (b) Trellis for $(1+D)^{-1}$-precoded rate $2/3$ $(1,7)$ W&W encoder.

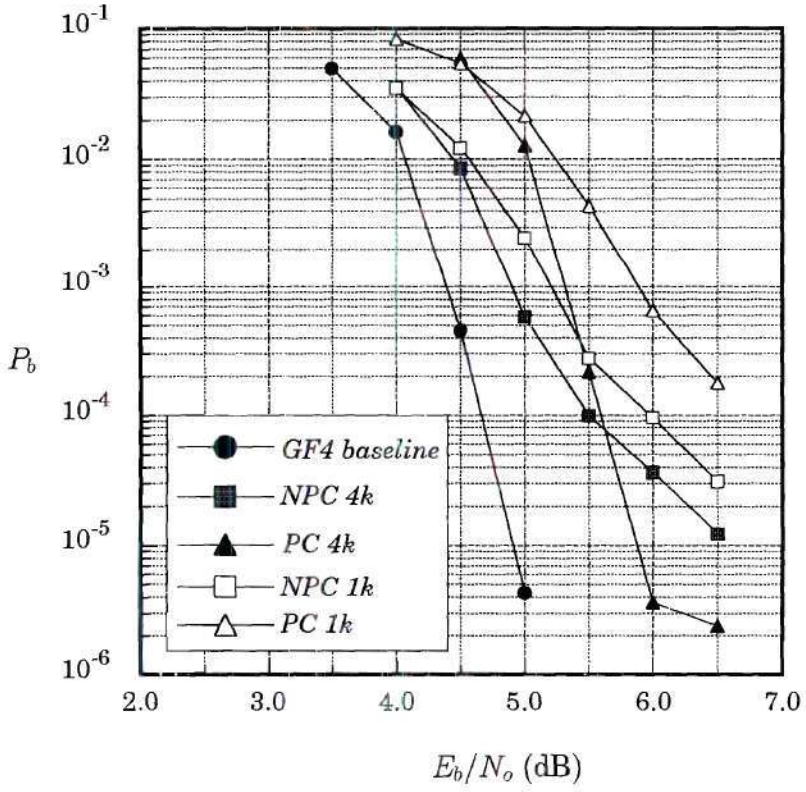


Figure 6.12. Serial concatenation of an outer rate 8/9 (31, 33) RSC and rate 2/3 (1, 7) W&W inner encoder in AWGN. Precoder is GF(4) $1/(1+D)$. 5 iterations. Interleaver size: 1000 and 4000. "GF(4) baseline" is inner $1/(1+D)$ precoder only with a size-1000 interleaver. "NPC" indicates "no precoder."

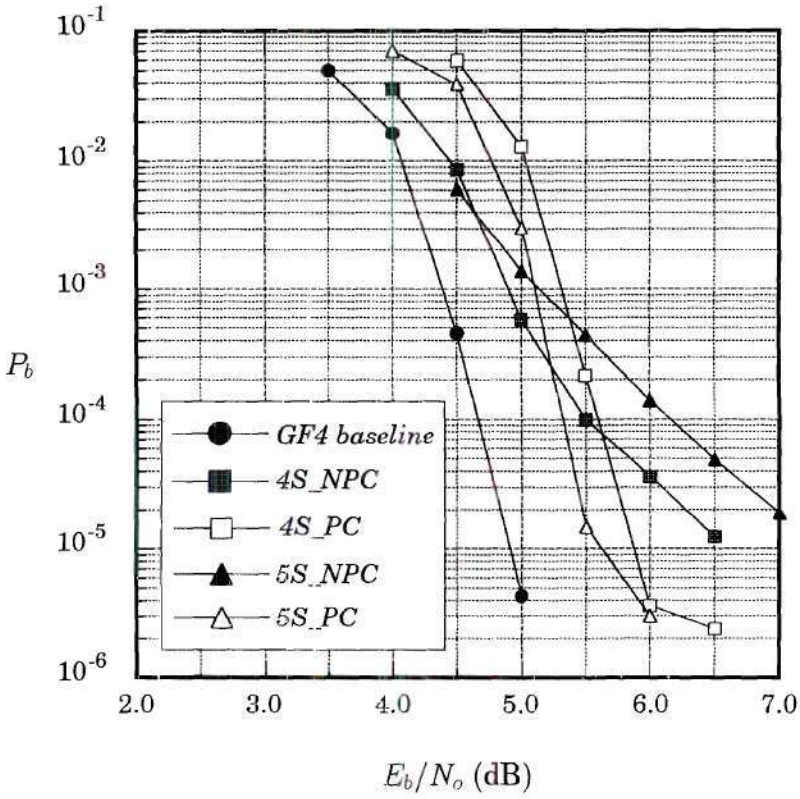


Figure 6.13. Serial concatenation of an outer rate 8/9 (31, 33) RSC and rate 2/3 (1, 7) inner encoders in AWGN. Interleaver size is 4000. "4S" refers to the W&W encoder, and "5S" to the AHM encoder.

The price for this is a small complexity increase by way of an extra state in the precoded trellis.

CHAPTER VII

CONCLUDING REMARKS

7.1 Summary of Results

This research has had as its chief goal the identification, analysis, and evaluation of coded digital communication systems that allow a constrained code to be used in conjunction with an interleaved concatenated code. An important property of all of these systems is that they should possess the ability to recover a portion of the rate loss of the constrained code. Of particular practical interest have been those systems that allow for high rate loss recovery with modest attendant increases in receiver complexity. The research addressed the following issues:

1. Viability and optimal decoding of a $(0, k)$-constrained-input turbo-coded system.
2. Analysis of the performance limits of high-rate $(0, k)$-constrained block codes in constrained AWGN channels.
3. Evaluation of systems of high-rate $(0, k)$-constrained block codes cascaded with turbo codes in constrained AWGN and PR channels.
4. Determination of how to obtain reliable soft information for an iterative ECC decoder from nonlinear block code decoders.

5. Determination of how to deliver a (d, k)-constrained sequence to a channel within an interleaved serial concatenated framework.

Contributions from this research are as follows:

1. Optimal decoding of a system of $(0, k)$-constrained-input turbo codes has been demonstrated. System characteristics and performance limits have been obtained.
2. Performance limits, in terms of BER, for several families of block rate $(n-1)/n$ $(0, k)$-constrained codes in constrained AWGN channels have been obtained.
3. Construction of, and encoders for code families that achieve these BER performance limits has been shown.
4. A method of correctly implementing a SISO decoder for an arbitrary block code has been demonstrated. The complexity of this algorithm has been determined and compared with that for a trellis-based decoder.
5. It has been demonstrated that systematic block $(0, k)$-constrained codes can be used in several interleaved serial concatenated systems with near-zero rate loss.
6. A general framework for interleaved serial concatenated precoded (d, k)-constrained systems in constrained AWGN channels has been shown.

7.1.1 $(0, k)$-Constrained-Input Turbo Codes

Since a turbo encoder is systematic, this 'reverse concatenation' system allows the soft-input turbo decoder to obtain reliable soft information directly from the channel

detector. It has been demonstrated that operating the BCJR algorithm over Cartesian product trellises of RSC trellises and $(0, k)$-constraint FSTDs can recover varying amounts of the rate loss for arbitrary $(0, k)$ codes. The optimal combinations are $k \leq 4$ and high-efficiency constrained codes. Net coding gain of a few tenths of a dB has been shown to be possible.

7.1.2 Performance of High-Rate $(0, k)$ Block Codes

It has been demonstrated that block $(0, k)$-constrained codes of rate $(n-1)/n$ with $d_{min} = 1$ can be constructed from a 'mother' code which is the odd SPC code of length n. Despite the fact that they have $d_{min} = 1$, it has been shown that, with an ML or MAP decoder in AWGN, these codes are capable of coding gain or very high rate loss recovery. Construction of several code families has shown a fundamental tradeoff between BER performance and the runlength k parameter—a weaker code accompanying smaller k. The construction of these code families has been used in concert with an ML union bound for nonlinear block codes to establish BER performance limits.

7.1.3 Turbo Codes Cascaded with High-Rate Block Codes on $(0, k)$-Constrained Channels

In conjunction with the standard recording system configuration of an outer ECC/inner constrained code, a SISO algorithm for a general block code has been proposed for obtaining soft information from the decoder for a short high-rate $(0, k)$-constrained block code. The implementation complexity of this algorithm has been documented and compared with that for the BCJR algorithm. The soft information is delivered to a soft-input iterative decoder for an outer turbo code. Simulation results indicate that complete rate loss recovery (and modest coding gains) are

possible on constrained AWGN channels. The attractiveness of systematic $(0, k)$ modulation codes for PR (ISI in general) channels, in terms of rate loss recovery and implementation complexity, has also been proposed. The fixed coordinates of these systematic codes, although not contributing any additional distance to the code, have been shown to be effective in improving the performance of an APP algorithm permitting good rate loss recovery of the constrained code on PR channels.

7.1.4 (d, k)-Constrained Serial Concatenation

Several systems for incorporating (d, k) codes into an interleaved serial concatenation system have been proposed. These systems involve systematic modulation codes and several examples of finite-state constrained encoders. A system that makes use of systematic $(0, k)$ modulation codes and fully exploits the fixed coordinates of these codes to achieve near-zero rate loss of the modulation code has been proposed and noted for its simplicity and effectiveness. A general way to employ recursive precoders defined over binary extension fields GF(2^p) to obtain interleaver gain for rate p/q modulation codes with finite-state encoders has also been described.

7.2 Suggestions for Future Research

7.2.1 Constrained-Input Turbo Codes II

The results on constrained-input turbo codes in Section 3.1.4 indicate that substantial coding gains could be achieved when a turbo encoder is driven with sequences from a subset of the $(0, k)$ constraints—the $X_{10^{(j)}1}(0, k)$ constraints—if the inputs to *both* RSC encoders can be made to satisfy the constraint. Then a $(0, k)$ constraint FSTD $\times$ RSC Cartesian product trellis can be utilized in each APP

detector for the turbo code. The system used to demonstrate this possibility made use of an impractical time-varying random interleaver. A practical input-constrained turbo-coded system admitting a static interleaver is shown in Figure 7.1.

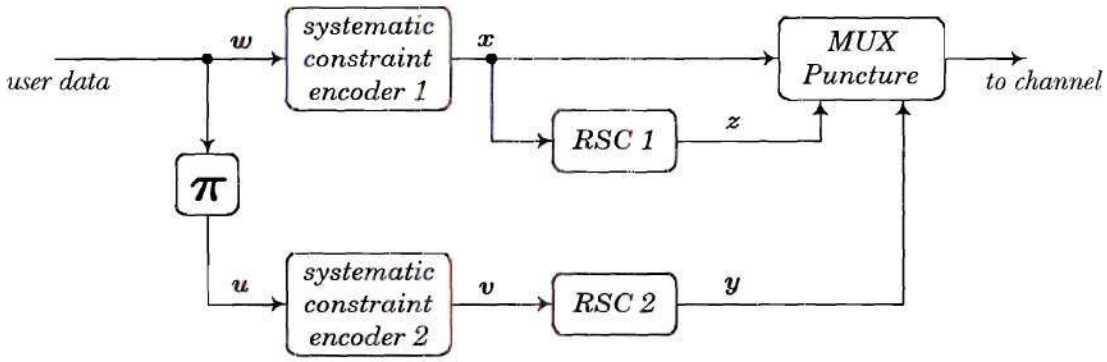


Figure 7.1. Constrained-input turbo coded system permitting the use of a $(0, k)$ constraint FSTD $\times$ RSC Cartesian product trellis in each APP detector.

This system will deliver a $(0, k)$-constrained sequence to the channel. The passage of extrinsic information $L(\cdot)$ within the decoder for this system is as follows. APP 1, operating over a joint trellis, computes $L(\boldsymbol{x})$ from the received noisy modulated versions of $\boldsymbol{x}$ and $\boldsymbol{z}$ (these are available after the channel output has been demultiplexed); the SISO decoder for constraint encoder 1 (SISO 1) then obtains $L(\boldsymbol{w})$ from $L(\boldsymbol{x})$ by discarding the LAPPs for the constraint parity bits. Then $L^{apr}(\boldsymbol{u})$ is obtained as $L^{apr}(\boldsymbol{u}) = \boldsymbol{\pi}(L(\boldsymbol{w}))$ after which the SISO decoder for constraint encoder 2 (SISO 2) obtains $L^{apr}(\boldsymbol{v})$ from $L^{apr}(\boldsymbol{u})$ by inserting the appropriate *a priori* values of $\pm\infty$ for the constraint parity bits into $L^{apr}(\boldsymbol{u})$. APP 2, also operating over a joint trellis, computes $L(\boldsymbol{v})$ from the received noisy modulated versions of $\boldsymbol{x}$ and $\boldsymbol{y}$ using $L^{apr}(\boldsymbol{v})$ as *a priori* on $\boldsymbol{v}$. SISO 2 then extracts $L(\boldsymbol{u})$ from $L(\boldsymbol{v})$ by removal of the LLRs

for the constraint parity bits. This completes an iteration and, for BPSK modulation, a decision on $\boldsymbol{w}$ can now be made using

$$\widehat{\boldsymbol{w}} = sgn\left[\pi^{-1}(L(\boldsymbol{u})) + L(\boldsymbol{w}) + L_c \cdot M(\boldsymbol{w})\right]$$

with $M(\boldsymbol{w})$ representing the received noisy modulated version of $\boldsymbol{w}$ — which can be obtained from the channel output: L_c is the channel reliability function.

This system requires a systematic block code for the $X_{10(0)1}(0,k)$ constraints in order to avoid severe error propagation when $L(\boldsymbol{v})$ is computed from $L(\boldsymbol{u})$ — or $L(\boldsymbol{x})$ from $L(\boldsymbol{w})$. It would be interesting to determine if systematic block codes of sufficiently high efficiency exist for these constraints, and how much rate loss recovery or coding gain can be achieved in this system with their use. Of greater interest would be description of an equivalent practical system that does not rely on a systematic code.

7.2.2 (d,k)-Constraints on Partial Response Channels

The $(1,k)$ constraint, when passed through a $1/(1 \oplus D)$ precoder, is known to provide coding gain on some PR channels. Of interest would be description of systems that extend the (d,k)-constrained serial concatenation of Section 6.2 to PR channels. More generally, high-efficiency finite-state encoders exist for distance-enhancing constraints for high-order PR channels [39]; these encoders could be mated with PR channels within this serial concatenation framework. This same framework can also be expanded to encompass the application of MSN codes to PR channels.

APPENDIX A

CONSTRUCTION OF $\mathcal{C}_n''$, $\mathcal{C}_n^{4\prime}$ AND THEIR ENCODERS

A.1 Construction of $\mathcal{C}_n''$ and its Encoder

Let us replace two codewords in $\mathcal{C}_n$ instead of just one; in particular, let the replaced codewords be

$$\begin{aligned} \boldsymbol{c}_{r1} &= 0^{(n-1)}1 \\ \boldsymbol{c}_{r2} &= 10^{(n-1)}. \end{aligned} \tag{A.1}$$

We then have the $(0, 2n-4)$ code $\mathcal{C}_n''$ with the least number of $d_H = 1$ codeword pairs. In order to evaluate β_1 for this case we need to know the minimum possible number of $d_H = 1$ codeword pairs in $\mathcal{C}_n''$. The following two facts give the required minimum.

Fact A.1. There are exactly two n-tuples that are $d_H = 1$ from both $\boldsymbol{d}_1$and $\boldsymbol{d}_2$; $\boldsymbol{d}_1$and $\boldsymbol{d}_2$ being distinct identical-parity n-tuples.

Proof: Since $\boldsymbol{d}_1$and $\boldsymbol{d}_2$ have identical parity they are a minimum of $d_H = 2$ apart; they therefore differ in i coordinates, where $i \in \{2, 4, 6, \cdots, m\}$, where m is the largest even number $\leq n$. Suppose $d_H(\boldsymbol{d}_1, \boldsymbol{d}_2) = i$; since $\boldsymbol{d}_1$and $\boldsymbol{d}_2$ agree in $n - i$ coordinates, we may, without loss of generality, represent them by

$$\begin{aligned} \boldsymbol{d}_1 &= n_1, n_2, \cdots, n_i, y_1, y_2, \cdots, y_{n-i} \\ \boldsymbol{d}_2 &= \overline{n_1}, \overline{n_2}, \cdots, \overline{n_i}, y_1, y_2, \cdots, y_{n-i} \end{aligned} \tag{A.2}$$

with the 'y's being the $n-i$ bits where $\boldsymbol{d}_1$and $\boldsymbol{d}_2$ agree and the 'n's the i bits where they disagree. There are three cases to consider.

Case (*i*): Let $\boldsymbol{x}_1$, $d_H = 1$ from $\boldsymbol{d}_1$, be obtained by complementing one of the common bits y_j of $\boldsymbol{d}_1$; let $\boldsymbol{x}_2$, $d_H = 1$ from $\boldsymbol{d}_2$ also be obtained by complementing one of the common bits y_k of $\boldsymbol{d}_2$. Clearly, $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ must differ in at least i coordinates (exactly i when $j = k$); so they cannot be identical.

Case (*ii*): Let $\boldsymbol{x}_1$, $d_H = 1$ from $\boldsymbol{d}_1$, be obtained by complementing one of the common bits y_j of $\boldsymbol{d}_1$; let $\boldsymbol{x}_2$, $d_H = 1$ from $\boldsymbol{d}_2$ be obtained by complementing one of the non-common bits n_k of $\boldsymbol{d}_2$. Here also, since $d_H(\boldsymbol{x}_1, \boldsymbol{x}_2) = i$ (still), $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ cannot be identical. This argument is obviously symmetric in $\boldsymbol{d}_1$and $\boldsymbol{d}_2$.

Case (*iii*); Let $\boldsymbol{x}_1$, $d_H = 1$ from $\boldsymbol{d}_1$, be obtained by complementing one of the non-common bits n_j of $\boldsymbol{d}_1$; let $\boldsymbol{x}_2$, $d_H = 1$ from $\boldsymbol{d}_2$ also be obtained by complementing one of the non-common coordinates n_k of $\boldsymbol{d}_2$. For this case, $d_H(\boldsymbol{x}_1, \boldsymbol{x}_2) \geq i - 2$ (equality when $j \neq k$); so they cannot be identical unless $i = 2$.

So in order to find all the n-tuples that are $d_H = 1$ from both $\boldsymbol{d}_1$and $\boldsymbol{d}_2$, we need concentrate only on Case (*iii*). For Case (*iii*), if $4 \leq i \leq m$ there are no n-tuples $d_H = 1$ from both $\boldsymbol{d}_1$and $\boldsymbol{d}_2$; if $i = 2$ there are only 2 choices for either $\boldsymbol{x}_1$ or $\boldsymbol{x}_2$—in both cases, both $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are $d_H = 1$ from both $\boldsymbol{d}_1$and $\boldsymbol{d}_2$. So the only number of such n-tuples is 2. □

Fact A.2. The minimum number of $d_H = 1$ codeword pairs in $\mathcal{C}''_n$ is $2n - 3$.

Proof: Let $\boldsymbol{c}_{r1}$ be replaced with $\boldsymbol{c}''_{e1}$to obtain, as an intermediate step, $\mathcal{C}'_n$; from Fact 4.4 there can be as few as $n - 1$ codewords in $\mathcal{C}'_n$that are $d_H = 1$ from $\boldsymbol{c}''_{e1}$. Next we

replace $\boldsymbol{c}_{r2}$ with $\boldsymbol{c}''_{e2}$ to obtain $\mathcal{C}''_n$. There can now be as few as $n-2$ odd-weight codewords $d_H = 1$ from $\boldsymbol{c}''_{e2}$ (both $\boldsymbol{c}_{r1}$ and $\boldsymbol{c}_{r2}$ could have been among the n codewords $d_H = 1$ from $\boldsymbol{c}''_{e2}$). Therefore, there could be as few as $(n-1)+(n-2)$ $d_H = 1$ codeword pairs in $\mathcal{C}''_n$. □

In order to continue, we need to be able to specify the exact patterns of the $d_H = 1$ codeword pairs in $\mathcal{C}''_n$. This is to ensure that the mapping of codewords to data blocks can actually be made such that the $d_H = 1$ codeword pairs in $\mathcal{C}''_n$ are mapped to $d_H = 1$ data blocks without conflict. So we argue for the following two assertions using some of the notation from Fact A.2:

- *Assertion 2.* $\boldsymbol{c}''_{e1}$ can always be chosen to replace $\boldsymbol{c}_{r1}$ so that $\boldsymbol{c}''_{e1}$ forms $n-1$ $d_H = 1$ codeword pairs of the form

$$\{(\boldsymbol{c}''_{e1}, \boldsymbol{a}_1), (\boldsymbol{c}''_{e1}, \boldsymbol{a}_2), \cdots, (\boldsymbol{c}''_{e1}, \boldsymbol{a}_{n-1})\} = \boldsymbol{A} \tag{A.3}$$

  where the $\boldsymbol{a}_i$ are distinct odd-parity codewords in $\mathcal{C}_n$. This case has already been discussed in Fact 4.4.

- *Assertion 3.* $\boldsymbol{c}''_{e2}$ can always be chosen to replace $\boldsymbol{c}_{r2}$ so that $\boldsymbol{c}''_{e2}$ forms $n-2$ $d_H = 1$ codeword pairs of the form

$$\{(\boldsymbol{c}''_{e2}, \boldsymbol{a}_1), (\boldsymbol{c}''_{e2}, \boldsymbol{b}_1), (\boldsymbol{c}''_{e2}, \boldsymbol{b}_2), \cdots, (\boldsymbol{c}''_{e2}, \boldsymbol{b}_{n-3})\} = \boldsymbol{B} \tag{A.4}$$

  where the $\boldsymbol{b}_i$ are also distinct odd-parity codewords in $\mathcal{C}_n$, with

$$b_j \neq a_k, \forall\, j \in \{1, 2, \cdots, n-3\}, k \in \{1, 2, \cdots, n-1\}$$

  Without any loss of generality, we have indicated the common $d_H = 1$ neighbor to $\boldsymbol{c}''_{e1}$and $\boldsymbol{c}''_{e2}$ as $\boldsymbol{a}_1$. So $\boldsymbol{a}_1 = 10^{(n-3)}11$.

We observe that since we intend to set $\boldsymbol{c}_{r1} = 0^{(n-1)}1$ and $\boldsymbol{c}_{r2} = 10^{(n-1)}$, we can always pick $\boldsymbol{c}''_{e2} = 10^{(n-2)}1$ since it is $d_H = 1$ from both $\boldsymbol{c}_{r1}$ and $\boldsymbol{c}_{r2}$ (recall we want to minimize the number of $d_H = 1$ pairs in $\mathcal{C}''_n$); so $\boldsymbol{c}''_{e2}$ will have only $n-2$ $d_H = 1$ neighbors in $\mathcal{C}''_n$. Since the only other codeword in $\mathcal{C}^*_n$ (the even-weight n-tuples) that is $d_H = 1$ from both $0^{(n-1)}1$ and $10^{(n-1)}$ is $0^{(n)}$ (which we will not pick since we need to satisfy $(0, k)$ constraints), it follows from Fact A.1 that any other $\boldsymbol{c}''_{e1} \neq 0^{(n)}$ that we pick cannot have both $10^{(n-1)}$ and $0^{(n-1)}1$ as $d_H = 1$ neighbors. Therefore $\boldsymbol{c}''_{e1}$ and $\boldsymbol{c}''_{e2}$ can have only one common $d_H = 1$ neighbor as asserted. An obvious choice for $\boldsymbol{c}''_{e1}$ is $0^{(n-2)}11$ since it is $d_H = 1$ from $\boldsymbol{c}_{r1} = 0^{(n-1)}1$. To recapitulate, the two codewords

$$\begin{aligned} \boldsymbol{c}_{r1} &= 0^{(n-1)}1 \\ \boldsymbol{c}_{r2} &= 10^{(n-1)} \end{aligned} \tag{A.5}$$

in $\mathcal{C}_n$ are replaced with

$$\begin{aligned} \boldsymbol{c}''_{e1} &= 0^{(n-2)}11 \\ \boldsymbol{c}''_{e2} &= 10^{(n-2)}1 \end{aligned} \tag{A.6}$$

respectively, to obtain $\mathcal{C}''_n$.

We must next show that there exist data block $d_H = 1$ pair sets similar to $\boldsymbol{A}$ and $\boldsymbol{B}$ above for the $d_H = 1$ codeword pairs. Let $\boldsymbol{g}$ be an arbitrary $(n-1)$-tuple; the existence of a set identical to $\boldsymbol{A}$ in (A.3) is obviously guaranteed. From Fact A.1, any other $(n-1)$-tuple $\boldsymbol{h}$ $d_H = 2$ from $\boldsymbol{g}$ has exactly two $d_H = 1$ neighbors in common with $\boldsymbol{g}$. Therefore the existence of the 2 sets of $d_H = 1$ $(n-1)$-tuple pairs of the form

$$\begin{aligned} \{(\boldsymbol{g}, \boldsymbol{c}_1), (\boldsymbol{g}, \boldsymbol{c}_2), (\boldsymbol{g}, \boldsymbol{c}_3), (\boldsymbol{g}, \boldsymbol{c}_4), \cdots, (\boldsymbol{g}, \boldsymbol{c}_{n-1})\} &= \boldsymbol{C}_{dw} \\ \{(\boldsymbol{h}, \boldsymbol{c}_1), (\boldsymbol{h}, \boldsymbol{c}_2), (\boldsymbol{h}, \boldsymbol{d}_1), (\boldsymbol{h}, \boldsymbol{d}_2), \cdots, (\boldsymbol{h}, \boldsymbol{d}_{n-3})\} &= \boldsymbol{D}_{dw} \end{aligned} \tag{A.7}$$

is established for $n \geq 3$; and $d_H(\boldsymbol{g}, \boldsymbol{h}) = 2$. The data block-to-codeword assignments consistent with the sets $\boldsymbol{A}$ and $\boldsymbol{C}_{dw}$ is obvious; regarding sets $\boldsymbol{B}$ and $\boldsymbol{D}_{dw}$, we can make the assignment

$$\boldsymbol{h} \mapsto \boldsymbol{c}''_{e2}, \boldsymbol{b}_1 \mapsto \boldsymbol{d}_1, \boldsymbol{b}_2 \mapsto \boldsymbol{d}_2, \cdots, \boldsymbol{b}_{n-3} \mapsto \boldsymbol{d}_{n-3}. \tag{A.8}$$

This assignment forces every $d_H = 1$ codeword pair to correspond to a $d_H = 1$ data block pair; it is not unique, however.[15] As an example for $n = 4$, we have

$$C_4 = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \begin{matrix} \leftarrow \boldsymbol{c}_{r1} \\ \\ \\ \\ \leftarrow \boldsymbol{c}_{r2} \\ \\ \\ \\ \end{matrix} \Longrightarrow C''_4 = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{1} \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{1} \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \begin{matrix} \leftarrow \boldsymbol{c}''_{e1} \\ \\ \\ \\ \leftarrow \boldsymbol{c}''_{e2} \\ \\ \\ \\ \end{matrix}$$

The codeword $d_H = 1$ pairs are (as listed in the rows of the matrix C''_4)

$$\begin{aligned} \boldsymbol{A} &= \{(0,1),(0,3),(0,5)\}. \\ \boldsymbol{B} &= \{(4,5),(4,6)\} \end{aligned}$$

It is a simple matter to verify that two viable sets of $d_H = 1$ 3-tuples (data blocks) are

$$\begin{aligned} \boldsymbol{C}_{dw} &= \{(0,1),(0,2),(0,4)\} \\ \boldsymbol{D}_{dw} &= \{(3,1),(3,2),(3,7)\} \end{aligned}$$

given in the decimal value of the lexicographic binary ordering. The unlisted 3-tuples (5 and 6) are unconstrained and are not critical in the data word-to-codeword mapping. We reiterate that the essence of the optimal mapping is in the *patterns*, and not the actual data blocks. Note, for instance, that the two common 3-tuples in $\boldsymbol{C}_{dw}$ and $\boldsymbol{D}_{dw}$ are 0 and 3, respectively ('headers'). The appearance of exactly two common $d_H = 1$ neighbors (1 and 2) of these headers is because of the fact that $d_H(000, 011) = 2$ (Fact A.1). Since there are a total of $\binom{3}{2} = 3$ pairs like this given a particular header for set $\boldsymbol{C}_{dw}$, there are conditionally 3 possible headers for the set

---

[15] $\boldsymbol{h} \mapsto \boldsymbol{c}''_{e2}, \boldsymbol{b}_2 \mapsto \boldsymbol{d}_1, \boldsymbol{b}_3 \mapsto \boldsymbol{d}_2, \cdots, \boldsymbol{b}_{n-3} \mapsto \boldsymbol{d}_{n-4}, \boldsymbol{b}_1 \mapsto \boldsymbol{d}_{n-3}$, for example, would also work.

D_{dw}. But since the distance between the data block pairs in C_{dw} and D_{dw} is unaffected by the addition of an arbitrary 3-tuple to every data block, there are a total of $2^3 = 8$ choices for the header for C_{dw}. There are therefore a total of 24 acceptable permutations of the pair $\{C_{dw}, D_{dw}\}$. This is easily generalized to a total of $\binom{n-1}{2}2^{n-1}$ acceptable pairs $\{C_{dw}, D_{dw}\}$ for codeword length n — a rather sizable number. This large number allows quite a bit of design freedom in choosing a simpler encoder (perhaps by trial and error). So continuing with the example, the optimal mapping for *this particular* C_{dw} and D_{dw} would be

data block	*code word*	*data block*	*code word*
0 [0 0 0]	0 [0 0 1 1]	4 [1 0 0]	1 [0 0 1 0]
1 [0 0 1]	5 [1 0 1 1]	**5 [1 0 1]**	**2 [0 1 0 0]**
2 [0 1 0]	3 [0 1 1 1]	**6 [1 1 0]**	**7 [1 1 1 0]**
3 [0 1 1]	4 [1 0 0 1]	7 [1 1 1]	6 [1 1 0 1]

with the pairs set in bold being unconstrained (data 5 could be mapped to code word 7 and data 6 to code word 2 without compromise).

A.2 Construction of $\mathcal{C}_n^{4\prime}$ and its Encoder

For this case, the four codewords

$$\begin{aligned} c_{r1} &= 0^{(n-1)}1 \\ c_{r2} &= 10^{(n-1)} \\ c_{r3} &= 0^{(n-2)}10 \\ c_{r4} &= 010^{(n-2)} \end{aligned} \tag{A.9}$$

in $\mathcal{C}_n$ are replaced with

$$\begin{aligned} c_{e1}^{4\prime} &= 0^{(n-3)}101 \\ c_{e2}^{4\prime} &= 10^{(n-2)}1 \end{aligned} \tag{A.10}$$

$$\begin{aligned} \boldsymbol{c}_{e3}^{4\prime} &= 10^{(n-3)}10 \\ \boldsymbol{c}_{e4}^{4\prime} &= 010^{(n-4)}10 \end{aligned} \tag{A.10}$$

respectively. This set of $\{\boldsymbol{c}_{ei}^{4\prime}\}$ is not unique but it certainly produces a code $\mathcal{C}_n^{4\prime}$ with the least number of $d_H = 1$ codeword pairs. Since the goal is to minimize the number of $d_H = 1$ codeword pairs in the final code, we would like the $\{\boldsymbol{c}_{ei}^{4\prime}\}$ to have as many $d_H = 1$ neighbors in common (all of these neighbors are odd-parity n-tuples in $\mathcal{C}_n$, 4 of which are removed as the $\{\boldsymbol{c}_{ri}\}$). The rule then, is to select $\boldsymbol{c}_{ei}^{4\prime}$ so that two of its $d_H = 1$ neighbors are $\boldsymbol{c}_{ri}$ and $\boldsymbol{c}_{r(i-1)}$ while simultaneously satisfying the target $(0, 2n-6)$ constraint. Observe that in order for $\boldsymbol{c}_{e1}^{4\prime}$to have $\boldsymbol{c}_{r1}$ as a $d_H = 1$ neighbor, $\boldsymbol{c}_{e1}^{4\prime}$must be weight-2. If we pick $\boldsymbol{c}_{e1}^{4\prime}$as indicated above and if $\boldsymbol{c}_{e2}^{4\prime}$ must be $d_H = 1$ from both $\boldsymbol{c}_{r1}$ and $\boldsymbol{c}_{r2}$ then the only choice is $\boldsymbol{c}_{e2}^{4\prime} = \boldsymbol{c}_{r1} \oplus \boldsymbol{c}_{r2}$. So $\boldsymbol{c}_{e1}^{4\prime}$ and $\boldsymbol{c}_{e2}^{4\prime}$ have $n-1$ and $n-2$ $d_H = 1$ neighbors, respectively, in the partially constructed $\boldsymbol{c}_n^{4\prime}$. The common surviving $d_H = 1$ neighbor to both $\boldsymbol{c}_{e1}^{4\prime}$and $\boldsymbol{c}_{e2}^{4\prime}$ in the partially constructed $\boldsymbol{c}_n^{4\prime}$ is then $10^{(n-4)}101$. Now the only possible choice for $\boldsymbol{c}_{e3}^{4\prime}$ is $\boldsymbol{c}_{e3}^{4\prime} = \boldsymbol{c}_{r2} \oplus \boldsymbol{c}_{r3}$, and $\boldsymbol{c}_{e2}^{4\prime}$ and $\boldsymbol{c}_{e3}^{4\prime}$ share the common $d_H = 1$ neighbor $10^{(n-3)}10$. More generally, since all the $\{\boldsymbol{c}_{ri}\}$ are weight-1, exactly two of them may share a common $d_H = 1$ neighbor (we must exclude $0^{(n)}$). So continuing, $\boldsymbol{c}_{e4}^{4\prime}$ can be $d_H = 1$ from exactly two of the $\{\boldsymbol{c}_{ri}\}$. The only two choices are $\boldsymbol{c}_{e4}^{4\prime} = \boldsymbol{c}_{r1} \oplus \boldsymbol{c}_{r4}$ or $\boldsymbol{c}_{e4}^{4\prime} = \boldsymbol{c}_{r3} \oplus \boldsymbol{c}_{r4}$; we pick the latter since it allows more $d_H = 1$ matches to be made by the encoder. $\boldsymbol{c}_{e3}^{4\prime}$ and $\boldsymbol{c}_{e4}^{4\prime}$ then share the common $d_H = 1$ neighbor $110^{(n-4)}10$. So, the codeword $d_H = 1$ pairs are seen to be of the form,

$$\begin{aligned} \left\{\left(\boldsymbol{c}_{e1}^{4\prime}, \boldsymbol{w}_1\right), \left(\boldsymbol{c}_{e1}^{4\prime}, \boldsymbol{w}_2\right), \cdots, \left(\boldsymbol{c}_{e1}^{4\prime}, \boldsymbol{w}_{n-1}\right)\right\} &= \boldsymbol{A}_{cw} \\ \left\{\left(\boldsymbol{c}_{e2}^{4\prime}, \boldsymbol{w}_1\right), \left(\boldsymbol{c}_{e2}^{4\prime}, \boldsymbol{x}_1\right), \left(\boldsymbol{c}_{e2}^{4\prime}, \boldsymbol{x}_2\right), \cdots, \left(\boldsymbol{c}_{e2}^{4\prime}, \boldsymbol{x}_{n-3}\right)\right\} &= \boldsymbol{B}_{cw} \\ \left\{\left(\boldsymbol{c}_{e3}^{4\prime}, \boldsymbol{x}_1\right), \left(\boldsymbol{c}_{e3}^{4\prime}, \boldsymbol{y}_1\right), \left(\boldsymbol{c}_{e3}^{4\prime}, \boldsymbol{y}_2\right), \cdots, \left(\boldsymbol{c}_{e3}^{4\prime}, \boldsymbol{y}_{n-3}\right)\right\} &= \boldsymbol{C}_{cw} \\ \left\{\left(\boldsymbol{c}_{e4}^{4\prime}, \boldsymbol{y}_1\right), \left(\boldsymbol{c}_{e4}^{4\prime}, \boldsymbol{z}_1\right), \left(\boldsymbol{c}_{e4}^{4\prime}, \boldsymbol{z}_2\right), \cdots, \left(\boldsymbol{c}_{e4}^{4\prime}, \boldsymbol{z}_{n-3}\right)\right\} &= \boldsymbol{D}_{cw} \end{aligned} \tag{A.11}$$

giving a total of $4n-7$ $d_H = 1$ codeword pairs. From the foregoing discussion,

$$\begin{aligned} \boldsymbol{a}_1 &= 10^{(n-4)}101 \\ \boldsymbol{b}_1 &= 10^{(n-3)}11 \\ \boldsymbol{c}_1 &= 110^{(n-4)}10. \end{aligned} \tag{A.12}$$

Note that $\boldsymbol{c}_{e3}^{4\prime}$ and $\boldsymbol{c}_{e1}^{4\prime}$ cannot share any $d_H = 1$ neighbors since $d_H(\boldsymbol{c}_{e3}^{4\prime}, \boldsymbol{c}_{e1}^{4\prime}) = 4$; likewise, $d_H(\boldsymbol{c}_{e4}^{4\prime}, \boldsymbol{c}_{e1}^{4\prime}) = 4$, and $d_H(\boldsymbol{c}_{e4}^{4\prime}, \boldsymbol{c}_{e2}^{4\prime}) = 4$. This is consistent with the form of (A.11).

We now show that for $n \geq 6$ we can assemble sets of $(n-1)$-tuples that can accommodate the sets (A.11). We have already established the first two sets of $d_H = 1$ $(n-1)$-tuples of the form (A.7). Continuing with that notation, it is easily shown that, for an $(n-1)$-tuple $\boldsymbol{j}$, it is necessary and sufficient to have $d_H(\boldsymbol{g}, \boldsymbol{j}) = 4$ and $d_H(\boldsymbol{h}, \boldsymbol{j}) = 2$, simultaneously, in order to guarantee the existence of a set $\boldsymbol{E}_{dw}$ of the form

$$\{(\boldsymbol{j}, \boldsymbol{d}_1), (\boldsymbol{j}, \boldsymbol{d}_2), (\boldsymbol{j}, \boldsymbol{e}_1), (\boldsymbol{j}, \boldsymbol{e}_2), \cdots, (\boldsymbol{j}, \boldsymbol{e}_{n-3})\} = \boldsymbol{E}_{dw}. \tag{A.13}$$

Similarly, for an $(n-1)$-tuple $\boldsymbol{k}$, we must also have $d_H(\boldsymbol{g}, \boldsymbol{k}) = 4$, $d_H(\boldsymbol{h}, \boldsymbol{k}) = 4$, and $d_H(\boldsymbol{j}, \boldsymbol{k}) = 2$ simultaneously to guarantee a set $\boldsymbol{F}_{dw}$ of the form

$$\{(\boldsymbol{k}, \boldsymbol{e}_1), (\boldsymbol{k}, \boldsymbol{e}_2), (\boldsymbol{k}, \boldsymbol{f}_1), (\boldsymbol{k}, \boldsymbol{f}_2), \cdots, (\boldsymbol{k}, \boldsymbol{f}_{n-3})\} = \boldsymbol{F}_{dw}. \tag{A.14}$$

Without loss of generality, let the headers (i.e., $\boldsymbol{g}, \boldsymbol{h}, \boldsymbol{j}, \boldsymbol{k}$) for each of the sets $\boldsymbol{C}_{dw}$, $\boldsymbol{D}_{dw}$, $\boldsymbol{E}_{dw}$, and $\boldsymbol{F}_{dw}$ be even weight; then all the $\boldsymbol{c}_i, \boldsymbol{d}_i, \boldsymbol{e}_i$, and $\boldsymbol{f}_i$ are odd weight. Since the total number of odd-weight $(n-1)$-tuples is 2^{n-2} (= half of the total number of $(n-1)$-tuples), and

$$|\boldsymbol{C}_{dw}| + |\boldsymbol{D}_{dw}| + |\boldsymbol{E}_{dw}| + |\boldsymbol{F}_{dw}| = (n-1) + (n-3) + (n-3) + (n-3) = 4n - 10$$

the sets $\boldsymbol{C}_{dw}, \boldsymbol{D}_{dw}, \boldsymbol{E}_{dw}$, and $\boldsymbol{F}_{dw}$ are only all possible when

$$4n - 10 \leq 2^{n-2} \Rightarrow n \geq 6.$$

The assignment of data blocks to codewords can then be made as follows;

$$
\begin{aligned}
&\boldsymbol{g} \mapsto \boldsymbol{c}''_{e1}, \boldsymbol{c}_1 \mapsto \boldsymbol{w}_1, \boldsymbol{c}_2 \mapsto \boldsymbol{w}_2, \cdots, \boldsymbol{c}_{n-3} \mapsto \boldsymbol{w}_{n-3} \\
&\boldsymbol{h} \mapsto \boldsymbol{c}''_{e2}, \boldsymbol{d}_1 \mapsto \boldsymbol{x}_1, \boldsymbol{d}_2 \mapsto \boldsymbol{x}_2, \cdots, \boldsymbol{d}_{n-3} \mapsto \boldsymbol{x}_{n-3} \\
&\boldsymbol{j} \mapsto \boldsymbol{c}''_{e3}, \boldsymbol{e}_1 \mapsto \boldsymbol{y}_1, \boldsymbol{e}_2 \mapsto \boldsymbol{y}_2, \cdots, \boldsymbol{e}_{n-3} \mapsto \boldsymbol{y}_{n-3} \\
&\boldsymbol{k} \mapsto \boldsymbol{c}''_{e4}, \boldsymbol{f}_1 \mapsto \boldsymbol{z}_1, \boldsymbol{f}_2 \mapsto \boldsymbol{z}_2, \cdots, \boldsymbol{f}_{n-3} \mapsto \boldsymbol{z}_{n-3}.
\end{aligned}
\tag{A.15}
$$

For $n = 4$ and $n = 5$ we give examples (not unique but optimal) since they are not obvious from (A.15);

Table A.1. Optimal code/encoders for $\mathcal{C}_4^{4l}$ and $\mathcal{C}_5^{4l}$.

$\boldsymbol{d}$	$\mathcal{C}_4^{4l}(\boldsymbol{d})$	$\boldsymbol{d}$	$\mathcal{C}_4^{4l}(\boldsymbol{d})$
000	0101	100	1110
001	1101	101	1010
010	0111	110	0110
011	1001	111	1011

$\boldsymbol{d}$	$\mathcal{C}_5^{4l}(\boldsymbol{d})$	$\boldsymbol{d}$	$\mathcal{C}_5^{4l}(\boldsymbol{d})$	$\boldsymbol{d}$	$\mathcal{C}_5^{4l}(\boldsymbol{d})$	$\boldsymbol{d}$	$\mathcal{C}_5^{4l}(\boldsymbol{d})$
0000	00101	0100	00111	1000	01101	1100	11111
0001	10101	0101	01010	1001	01110	1101	11010
0010	00100	0110	01011	1010	11100	1110	10110
0011	10001	0111	10011	1011	11001	1111	10010

We have used this particular $\mathcal{C}_5^{4l}(\boldsymbol{d})$ as the $(0, 4)$ code in Sects. 4.3.3 and 5.2.

Bibliography

[1] R. L. Adler, D. Coppersmith, and M. Hassner, "Algorithms for sliding block codes. An application of symbolic dynamics to information theory," *IEEE Trans. Inform. Theory*, Vol. 29, No. 1, pp. 5–22, Jan. 1983.

[2] R. Adler, M. Hassner, and J. Moussouris, "Method and apparatus for generating a noiseless sliding block code for a $(1,7)$ channel with rate $2/3$," U.S. patent 4,413,251, 1983.

[3] K. D. Anim-Appiah and S. W. McLaughlin, "Constrained–input turbo codes for $(0,k)$ RLL channels," in *Proc., 1999 Conf. on Inform. Sciences and Systems,* pp. 420–425.

[4] K. D. Anim-Appiah and S. W. McLaughlin, "Towards soft output APP decoding for nonsystematic nonlinear block codes," in *Proc., 1999 Allerton Conf. on Commun., Control, and Computing,* Urbana, IL, pp. 1304–1313.

[5] K. Anim-Appiah and S. W. McLaughlin, "Turbo codes cascaded with high-rate block codes for $(0,k)$-constrained channels," to appear in *JSAC special issue on signal processing for high density storage channels.*

[6] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inform. Theory*, Vol. 20, pp. 284–287, March 1974.

[7] C. Berroux, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error correcting coding and decoding: Turbo codes," in *Proc., 1993 IEEE Int. Conf. Commun.*, pp. 1064–1070, 1993.

[8] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Serial concatenation of interleaved codes: Performance analysis, design, and iterative decoding," TDA progress report 42–126, JPL, Pasadena, CA, August 15, 1996.

[9] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "A soft-input soft-output maximum a posteriori (MAP) module to decode parallel and serial

concatenated codes," *TDA progress report* 42–127, JPL, Pasadena, CA, November 15, 1996.

[10] S. Benedetto and G. Montorsi, "Unveiling turbo codes: Some results on parallel concatenated coding," *IEEE Trans. Inform. Theory*, Vol. 42, No. 2, pp. 409–428, March 1996.

[11] S. Benedetto and G. Montorsi, "Average performance of parallel concatenated block codes," *Electron. Lett.*, Vol. 31, No. 3, pp. 156–158, Feb. 2, 1995.

[12] S. Benedetto and G. Montorsi, "Role of recursive convolutional codes in turbo codes," *Electron. Lett.*, Vol. 31, pp. 858–859, May 25, 1995.

[13] S. Benedetto and G. Montorsi, "Design of parallel concatenated codes," *IEEE Trans. Inform. Theory*, Vol. 44, No. 5, pp. 591–600, May 1996.

[14] E. Biglieri, D. Divsalar, P. J. McLane, and M. K. Simon, *Introduction to trellis-coded modulation with applications*. New York: Macmillan, 1991.

[15] W. G. Bliss, "Circuitry for performing error correction calculations on baseband encoded data to eliminate error propagation," *IBM Technol. Discl. Bull.*, Vol. 23, pp. 4633–4634, 1981.

[16] D. Chase, "A class of algorithms for decoding block codes with channel measurement information," *IEEE Trans. Inform. Theory*, Vol. IT-18, No. 1, pp. 170–182, Jan. 1972.

[17] T. M. Cover and J. A. Thomas, *Elements of information theory*. New York: Wiley and Sons, 1991.

[18] S. Datta and S. W. McLaughlin, "An Enumerative method for runlength limited codes: permutation codes," *IEEE Trans. Inform. Theory,* Vol. 45 No. 6, pp. 2199–2204, Sept. 1999.

[19] D. Divsalar, H. Jin, and R. J. McEliece, "Coding theorems for 'turbo-like' codes," in *Proc., 1998 Allerton Conf. on Commun., Control, and Computing*, Sept. 1998.

[20] D. Divsalar and F. Pollara, "Turbo codes for deep space communications," *TDA progress report* 42–120, JPL, Pasadena, CA, Feb. 15, 1995.

[21] J. Eggenberger and A. M. Patel, "Method and apparatus for implementing optimum PRML codes," *U. S. Patent 4,707,681*, Nov. 17, 1987.

[22] J. L. Fan, "Constrained coding and soft iterative decoding for storage," Ph.D. Dissertation, Stanford Univ., Stanford, CA, 2000.

[23] J. L. Fan and J. L. Cioffi, "Constrained coding techniques for soft decoders", in *Proc., 1999 Global Commun. Conf.,* 1999.

[24] J. Fitzpatrick and K. J. Knudson, "Rate 16/17 ($D = 0, G = 6/I = 7$) modulation codes for a magnetic recording channel," *U. S. Patent 5,635,933*, June. 3, 1997.

[25] G. D. Forney, Jr., *Concatenated codes.* Cambridge, Massachusetts: Massachusetts Institute of Technology, 1966.

[26] G. D. Forney, Jr., "Maximum-likelihood sequence estimation of digital sequences in the presence of intersymbol interference," *IEEE Trans. Inform. Theory*, Vol. IT-18, No. 3, pp. 363–378, May 1972.

[27] G. D. Forney, Jr. and M. V. Eyuboglu, "Combined equalization and coding using precoding," *IEEE Communcations Magazine*, Vol. 29, No.12, pp. 25–34, Dec. 1991.

[28] P. A. Franaszek,"Coding for constrained channels: a comparison of two approaches," *IBM J. Res. and Dev.*, Vol. 33, No. 6, pp. 602–608, Nov. 1989.

[29] J. Garcia-Frias and J. Villasenor, "Combining hidden Markov source models and parallel concatenated codes," *IEEE Commun. Lett.*, Vol. 1, No. 4, pp. 111–113, July 1997.

[30] J. Hagenauer, P. Hoeher, "A Viterbi algorithm with soft-decision outputs and its applications," in *Proc., 1989 IEEE Global Commun. Conf.,* pp. 1680–1686.

[31] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Trans. Inform. Theory*, Vol. 42, No. 2, pp. 429–445, March 1996.

[32] K. J. Hole and Ø. Ytrehus, "Improved coding techniques for precoded partial-response channels," *IEEE Trans. inform. Theory.*, Vol. 40, No. 2, pp. 482–493, Mar. 1994.

[33] K. A. S. Immink, "Runlength-limited sequences," *Proc. IEEE*, Vol. 78, pp. 1745–1759, Nov. 1990.

[34] K. A. S. Immink, P. H. Siegel, and J. K. Wolf, "Codes for digital recorders," *IEEE Trans. Inform. Theory*, Vol. 44, No. 6, pp. 2260–2299, Oct. 1998.

[35] J. Justesen, "Information rates and power spectra of digital codes," *IEEE Trans. Inform. Theory*, Vol. IT-28, No. 3, pp. 457–472, May 1982.

[36] P. Kabal and S. Pasupathy, "Partial-response signaling," *IEEE Trans. Commun.*, Vol. COM-23, No. 9, pp. 921–934, Sept. 1975.

[37] R. Karabed and P. H. Siegel, "Matched spectral null codes for partial response channels," *IEEE Trans. Inform. Theory*, Vol. 37, No. 3, pp. 818–855, May 1991.

[38] R. Karabed and P. H. Siegel, "Coding for higher order partial response channels," in *Proc. 1995 SPIE Int. Symp. Voice, Video, and Data Commun.*, Vol. 2605, pp. 115--125.

[39] R. Karabed and P. H. Siegel, "Constrained coding for binary channels with high intersymbol interference," *IEEE Trans. Inform. Theory*, Vol. 45 No. 6, pp. 1777–1797, Sept. 1999.

[40] H. Kobayashi, "Correlative level coding and maximum-likelihood decoding," *IEEE Trans. Inform. Theory*, Vol. IT-17, No. 5, pp. 586–594, Sept. 1971.

[41] H. Kobayashi, and D. T. Tang, "Application of partial-response channel coding to magnetic recording systems," *IBM J. Res. Dev.*, Vol. 14, pp. 368–375, July 1970.

[42] H. Koorapaty, Y.-P. E. Wang, and K. Balachandran, "Performance of turbo codes with short frame sizes," in *Proc. Veh. Technol. Conf.*, pp. 329–333, 1997.

[43] E. R. Kretzmer, "Generalization of a technique for binary data communication," *IEEE Trans. Commun. Technol. (Concise Papers)*, Vol. COM-14, pp. 67–68, Feb. 1966.

[44] A. Lender, "Correlative digital communication techniques," *IEEE Trans. Commun. Technol.*, Vol. COM-12, pp. 128–135, Dec. 1964.

[45] S. C. Li Ping and K. L. Yeung, "Iterative decoding of multi-dimensional concatenated single parity check codes," in *Proc., 1998 IEEE Int. Conf. Commun.*, pp. 131–135.

[46] B. H. Marcus, A. M. Patel, and P. H. Siegel, "Method and apparatus for implementing a PRML code," *U. S. Patent 4,786,890*, Nov. 22, 1988.

[47] B. Marcus, P. H. Siegel, and J. K. Wolf, "Finite state modulation codes for data storage," *IEEE J. Select. Areas Commun.*, Vol. 10 No. 1, pp. 5–37, Jan. 1992.

[48] S. W. McLaughlin, "Five runlength-limited codes for M-ary recording channels," *IEEE Trans. Magn.*, Vol. 33, No. 3, pp. 2442–2450, May 1997.

[49] S. W. McLaughlin, "Shedding light on the future of SP for optical recording," *IEEE Signal Processing Magazine*, pp. 83–94, July 1998.

[50] S. W. McLaughlin, P. Lee, R. Cloke, and B. Vasic, "One-pairs codes for partial response magnetic recording," *IEEE Trans. Inform. Theory.*, Vol. 35, Issue 3, Part 2, pp. 2080–2086, May 1999.

[51] L. McPheters, K. Narayanan, and S. McLaughlin, "Precoded PRML, serial concatenation, and iterative decoding for digital magnetic recording," *Inter. Conf. on Magn.*, May 1999.

[52] J. Moon, "The role of SP in data storage," *IEEE Signal Processing Magazine*, pp. 54–72, July 1998.

[53] J. Moon and B. Brickner, "Maximum transition run codes for data storage systems," *IEEE Trans. Magn.*, Vol. 32, No. 5, pp. 3992–3994, Sept. 1996.

[54] K. Narayanan and G. L. Stüber, "A Serial concatenation approach to iterative demodulation and decoding", *IEEE Trans. Commun.*, Vol. 47, Issue 7, pp. 956–961, July 1999.

[55] A. M. Patel, "Improved encoder and decoder for a byte-oriented rate $8/9$ $(0,3)$ code," *IBM Tech. Discl. Bull.*, Vol. 28, p. 1938, 1985.

[56] L. C. Perez, J. Seghers, and D. J. Costello Jr., "A distance spectrum interpretation of turbo codes," *IEEE Trans. Inform. Theory*, Vol. 42, No. 6, pp. 1698–1709, Nov. 1996.

[57] M. C. Reed and S. S. Pietrobon, "Turbo-code termination schemes and a novel alternative for short frames," *IEEE Intern. Symp. on Personal, Indoor and Mobile Radio Commun.*, Taipei, Taiwan, pp. 354–358, Oct. 1996.

[58] P. Robertson, "Illuminating the structure of code and decoder of parallel concatenated recursive systematic (turbo) codes," in *Proc., 1994 IEEE Global Commun. Conf.*, pp. 1298–1303, 1994.

[59] P. Robertson, E. Villebrun, and P. Hoeher, "A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain," in *Proc., 1995 IEEE Intern. Conf. Commun.*, pp. 1009–1013.

[60] W. E. Ryan, "A turbo code tutorial," unpublished paper. Available at *www.ee.virginia.edu/research/CCSP/turbo_codes/tcodesbib/turbo2c.ps.*

[61] W. Ryan, "Performance of high rate turbo codes on a PR4-equalized magnetic recording channel," in *Proc., 1998 IEEE Intern. Conf. Commun.*, pp. 947–951.

[62] W. Ryan, "Concatenated Codes for class IV partial response channels," to appear, *IEEE Trans. Comm.*

[63] W. E. Ryan, "Optimal code rates for concatenated codes on a PR4-equalized magnetic recording channel," in *Proc., 1999 Conf. Inform. Sciences and Systems,* Baltimore MD.

[64] W. E. Ryan, L. L. McPheters, and S. W. McLaughlin, "Combined turbo coding and turbo equalization for PR4-Equalized Lorentzian channels," in *Proc., 1998 Conf. Inform. Sciences and Systems,* pp. 489–494.

[65] E. Senata, *Nonnegative matrices and Markov Chains, 2nd ed.* New York: Springer-Verlag, 1981.

[66] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, Vol. 27, pp. 379–423, July 1948.

[67] D. Slepian, "Permutation modulation," *Proc. IEEE*, pp. 228–236, March 1965.

[68] T. Souvignier, A. Friedmann, M. Öberg, P. Siegel, R. E. Swanson, and J. K. Wolf, "Turbo decoding for PR4: parallel versus serial concatenation," *1999 IEEE Intern. Conf. Commun.*, Vancouver, British Columbia, June 1999.

[69] H. K. Thapar, and A. M. Patel, "A class of partial response systems for increasing storage density in magnetic recording," *IEEE Trans. Magn.*, Vol. MAG-23, No. 5, pp. 3666–3668, Sept. 1987.

[70] K. P. Tsang, "Method and apparatus for implementing run length limited codes in partial response channels," *U. S. Patent 5,537,112*, Jul. 16, 1996.

[71] B.F. Uchôa Filho and M. A. Herro, "Good convolutional codes for the precoded $(1-D)(1+D)^n$ partial-response channels", *IEEE Trans. Inform. Theory*, Vol. 43, No. 2, pp. 441–453, Mar. 1997.

[72] A. J. Viterbi and J. K. Omura, *Principles of digital communication and coding*. New York: McGraw-Hill, 1979.

[73] A. D. Weathers and J. K. Wolf, "A new rate 2/3 sliding block code for the $(1,7)$ runlength constraint with the minimal number of encoder states," *IEEE Trans. Inform. Theory*, Vol. 37, No. 3, pp. 908–913, May 1991.

[74] S. B. Wicker, *Error control systems for digital communication and storage*. Englewood Cliffs, NJ: Prentice-Hall, 1995.

[75] J. K. Wolf, "Permutation codes, (d,k) codes and magnetic recording," in *Proc. 1990 IEEE Colloquium in South America, Argentina, Brazil, Chile, Uruguay*, Sept. 1990. (Ed. W. Tompkins)

[76] J. K. Wolf and G. Ungerboeck, "Trellis coding for partial-response channels," *IEEE Trans. Commun.*, Vol. COM-34, pp. 765–773, Aug. 1986.

[77] E. Zehavi and J. K. Wolf, "On runlength codes," *IEEE Trans. Inform. Theory*, Vol. 34, No. 1, pp. 45–55, Jan. 1988.

Vita

Kofi D. Anim-Appiah was born in Kumasi, Ghana on April 28, 1967. He completed the GCE 'O' Level Examination in 1984 at Achimota School, Accra, Ghana, and received the International Baccalaureat (IB) diploma in 1986 from the International School of Tanganyika, Dar-es-Salaam, Tanzania. He subsequently received the Bachlor's and Master's degrees in Electrical Engineering in 1989 and 1991, respectively, from Virginia Polytechnic Institute & State University in Blacksburg, Virginia. He received the Ph.D. in Electrical Engineering in 2000 from the Georgia Institute of Technology in Atlanta, Georgia. From 1991 to 1995 he held a Design Engineering position at Eaton Corporation's Cutler-Hammer division in Milwaukee, Wisconsin where he was engaged in the design of inductive proximity sensors. His current research interests are in the general area of digital communications systems engineering with emphasis on error correction and modulation coding. He will be joining the Communications Systems Laboratory at Texas Instruments' DSP R&D Center in Dallas, Texas in December 2000.